Local Update Algorithms for Random Graphs

Romaric Duvignau

Journées ALÉA 2014

March 17, 2014

Joint Work with Philippe Duchon

Overview

- **Introduction**
- Update Algorithms
    - Deletion Algorithms
    - Insertion Algorithms
- Conclusion

General Setting

Context and Motivations

- Peer-to-Peer Network (structure maintained locally)
- Insertion and Deletion : **dependence** on the update sequence
- Malicious update sequence may perturb the network
- Difficulty in designing/analysing update algorithms
- Our suggestion : maintain **exactly** a given distribution for the network

General Setting

Context and Motivations

- Peer-to-Peer Network (structure maintained locally)
- Insertion and Deletion : **dependence** on the update sequence
- Malicious update sequence may perturb the network
- Difficulty in designing/analysing update algorithms
- Our suggestion : maintain **exactly** a given distribution for the network

Distribution-preserving update algorithms

- The network is modelled as a *random* graph G
- For each possible vertex set V, G should follow a given target distribution μ_V, which is *preserved* through updates :
  - **Insertion**: If $G \sim \mu_V$ and $u \notin V$ then $\mathcal{I}(G, u) \sim \mu_{V \cup \{u\}}$
  - **Deletion**: If $G \sim \mu_V$ and $u \in V$ then $\mathcal{D}(G, u) \sim \mu_{V \setminus \{u\}}$
- No probabilistic model for update sequences

Our graph model

k-out graphs

- Simple directed graphs with vertices of outdegree exactly k
- Good properties (low distances, etc) under the **uniform distribution** ν_V: All $N_G^+(v)$ are independent and each $N_G^+(v)$ is a uniform k-subset of $V - v$

Our graph model

k-out graphs

- Simple directed graphs with vertices of outdegree exactly k
- Good properties (low distances, etc) under the **uniform distribution** ν_V: All $N_G^+(v)$ are independent and each $N_G^+(v)$ is a uniform k-subset of $V - v$

Some properties of uniform n-vertices k-out graphs

- Indegrees follow the Binomial$(n-1, \frac{k}{n-1})$ distribution
- Connected with asymptotic probability 1

Our graph model

k-out graphs

- Simple directed graphs with vertices of outdegree exactly k
- Good properties (low distances, etc) under the **uniform distribution** ν_V: All $N_G^+(v)$ are independent and each $N_G^+(v)$ is a uniform k-subset of $V - v$

Some properties of uniform n-vertices k-out graphs

- Indegrees follow the Binomial$(n-1, \frac{k}{n-1})$ distribution
- Connected with asymptotic probability 1

Our goal: Local update algorithms

Given a uniform k-out graph G over V:

- **Deletion of u** $\in$ **V**: build a uniform k-out graph over $V \setminus \{u\}$
- **Insertion of u** $\notin$ **V**: build a uniform k-out graph over $V \cup \{u\}$

Our decentralized and cost model

Decentralized model

- Only use *local* knowledge and the *size* of the graph
- Access to a global primitive `RandomVertex()` that returns a uniform node of the vertex set

Our decentralized and cost model

Decentralized model

- Only use *local* knowledge and the *size* of the graph
- Access to a global primitive `RandomVertex()` that returns a uniform node of the vertex set

`RandomVertex()`: `RV` for short

- **Costly** primitive
- Cost of an update algorithm = expected number of calls to `RV`

Our decentralized and cost model

Decentralized model

- Only use *local* knowledge and the *size* of the graph
- Access to a global primitive `RandomVertex()` that returns a uniform node of the vertex set

`RandomVertex()`: `RV` for short

- **Costly** primitive
- Cost of an update algorithm = expected number of calls to `RV`

Our algorithms

- Minimize the symmetric difference between the input G and the output G'
- Constant expected time

Our results (best algorithms)

Optimal local update algorithms:

Deletion algorithm

- $o(1)$ calls to `RV`

Insertion algorithm

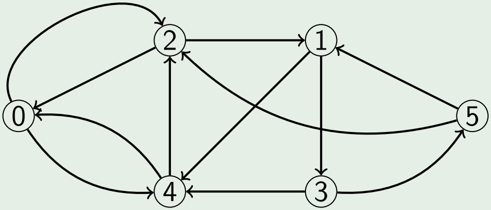
- k calls to `RV`

Overview

- Introduction
- Update Algorithms
  - **Deletion Algorithms**
  - Insertion Algorithms
- Conclusion

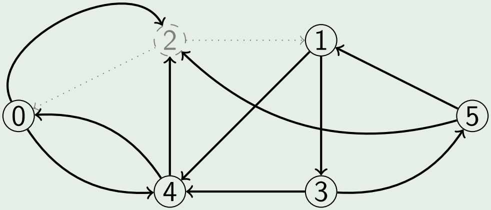
Deletion Algorithm: Simple Algorithm

Deletion of vertex 2



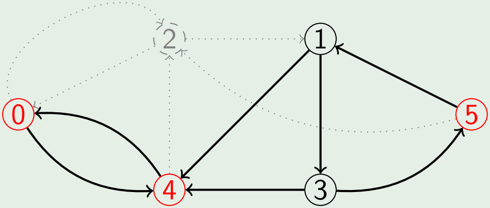
Deletion Algorithm: Simple Algorithm

Deletion of vertex 2



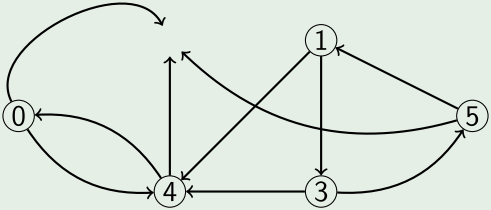
Deletion Algorithm: Simple Algorithm

Deletion of vertex 2



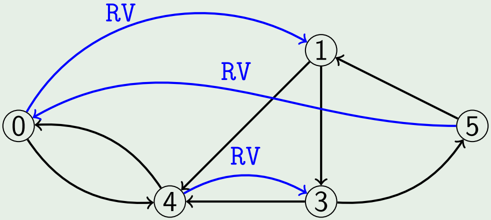
Deletion Algorithm: Simple Algorithm

Deletion of vertex 2



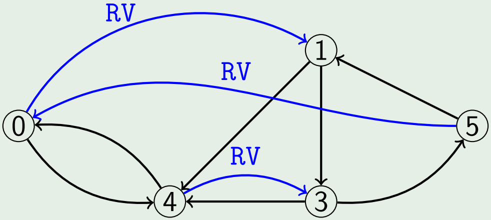
Deletion Algorithm: Simple Algorithm

Deletion of vertex 2



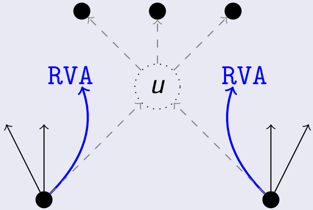
Deletion Algorithm: Simple Algorithm

Deletion of vertex 2



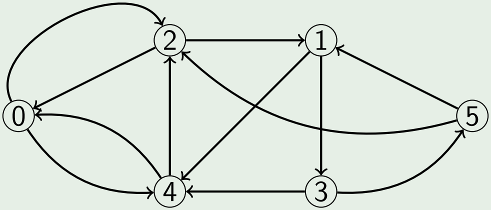
The *simple* solution

- Randomly redirect loose edges, avoiding incompatible choices
- Asymptotic cost k



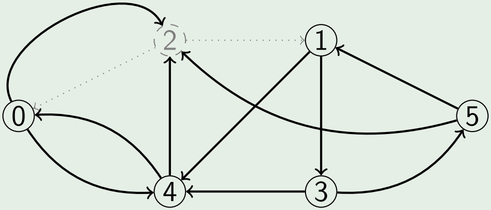
Deletion Algorithm: A better algorithm ?

Deletion of vertex 2



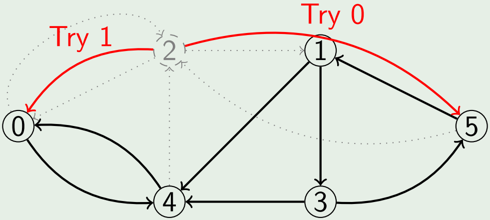
Deletion Algorithm: A better algorithm ?

Deletion of vertex 2



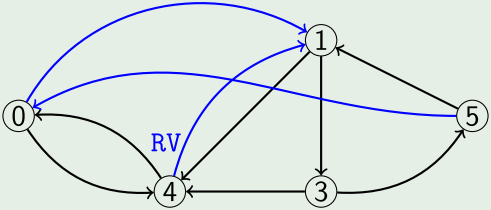
Deletion Algorithm: A better algorithm ?

Deletion of vertex 2



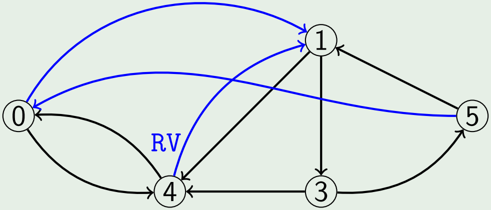
Deletion Algorithm: A better algorithm ?

Deletion of vertex 2



Deletion Algorithm: A better algorithm ?

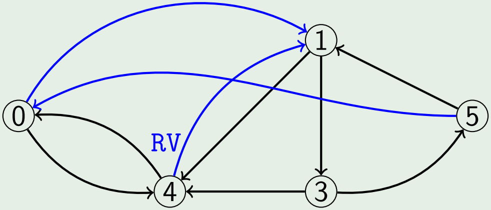
Deletion of vertex 2



- Suggestions must be independent, and can be made so

Deletion Algorithm: A better algorithm ?

Deletion of vertex 2



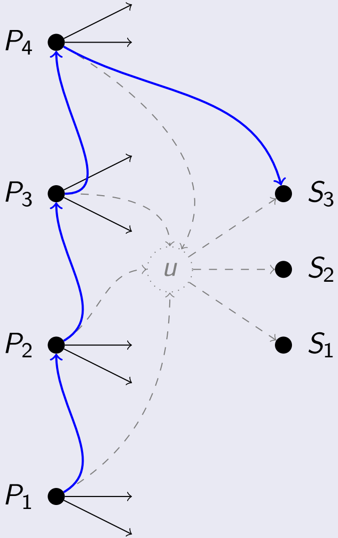
- Suggestions must be independent, and can be made so

Second algorithm

- Use $N^+(u)$ to save calls to RV, while preserving independence between suggestions
- Asymptotic Cost: $k \cdot \left(e^{-k} \cdot \frac{k^k}{k!}\right) \simeq \sqrt{\frac{k}{2\pi}}$

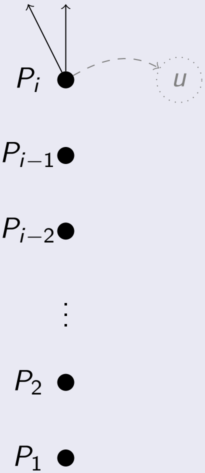
Deletion Algorithm: what about re-using $N^{-}(u)$?

Best Algorithm: the typical case



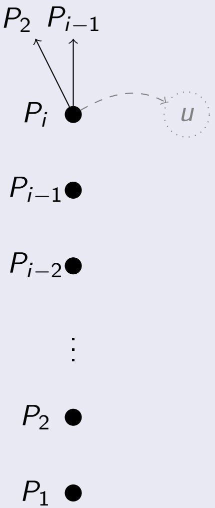
Deletion Algorithm: how to redirect loose edges ?

$L = |N_G^-(u)|$ and $1 \leq i \leq L-1$



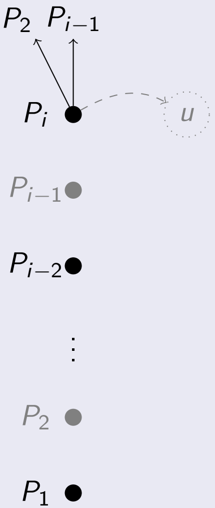
Deletion Algorithm: how to redirect loose edges ?

$L = |N_G^-(u)|$ and $1 \leq i \leq L - 1$



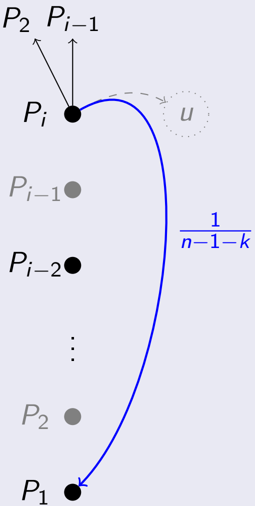
Deletion Algorithm: how to redirect loose edges ?

$L = |N_G^-(u)|$ and $1 \leq i \leq L - 1$



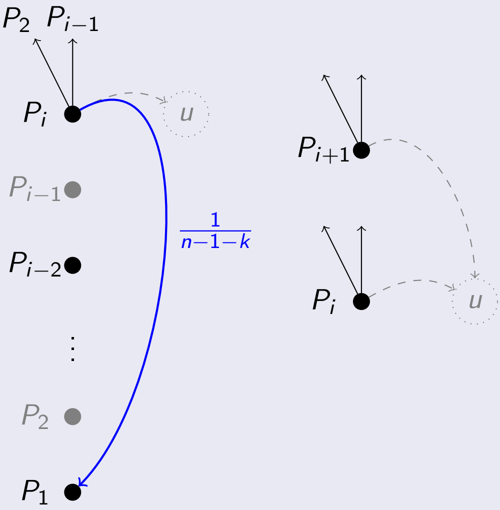
Deletion Algorithm: how to redirect loose edges ?

$L = |N_G^-(u)|$ and $1 \leq i \leq L-1$



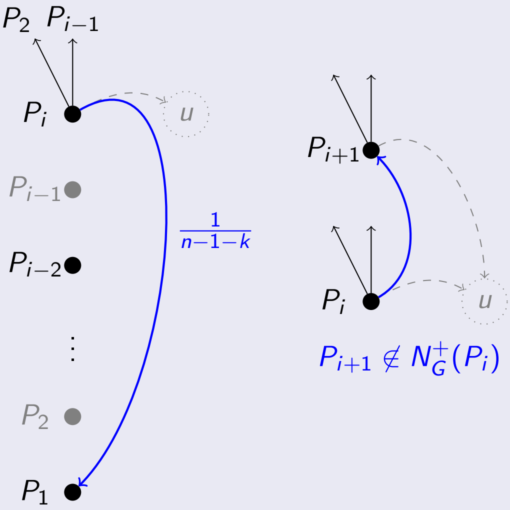
Deletion Algorithm: how to redirect loose edges ?

$L = |N_G^-(u)|$ and $1 \leq i \leq L-1$



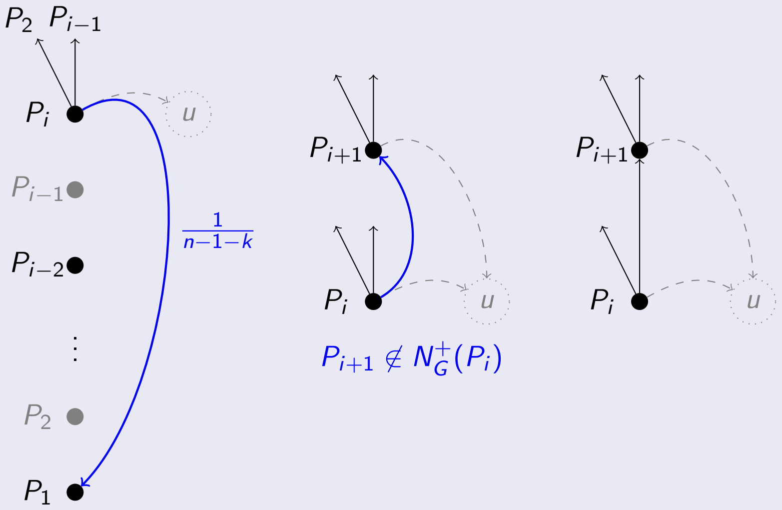
Deletion Algorithm: how to redirect loose edges ?

$L = |N_G^-(u)|$ and $1 \leq i \leq L-1$



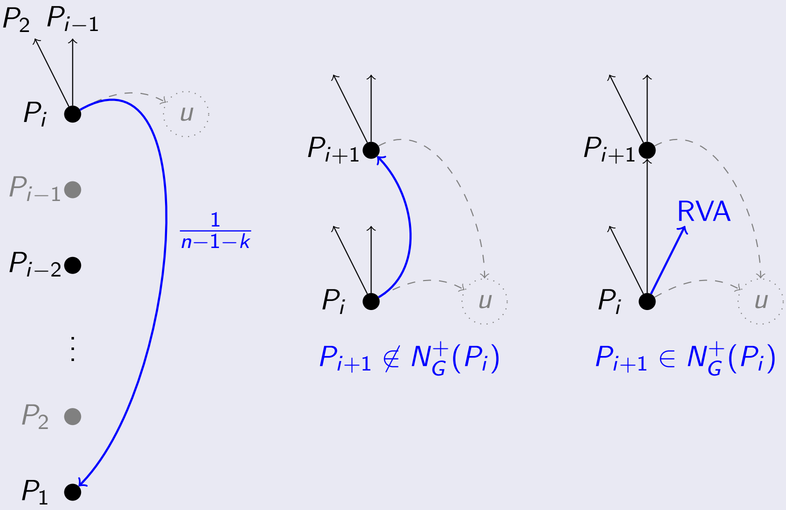
Deletion Algorithm: how to redirect loose edges ?

$L = |N_G^-(u)|$ and $1 \leq i \leq L-1$

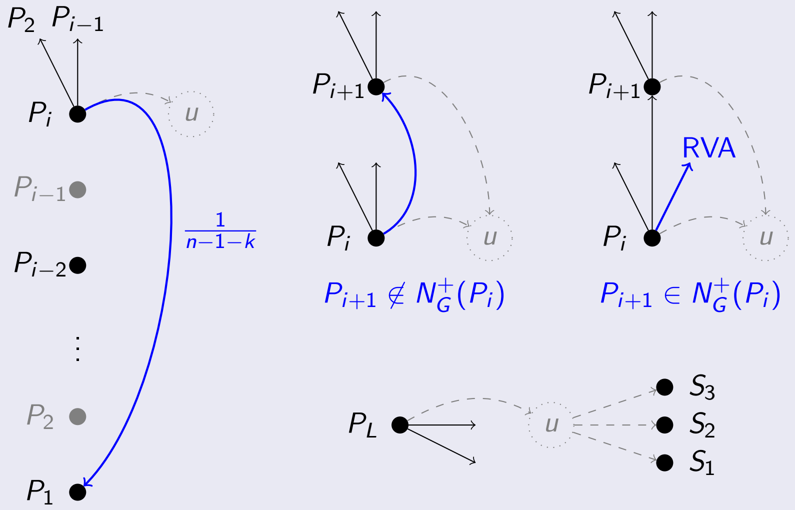


Deletion Algorithm: how to redirect loose edges ?

$L = |N_G^-(u)|$ and $1 \le i \le L-1$

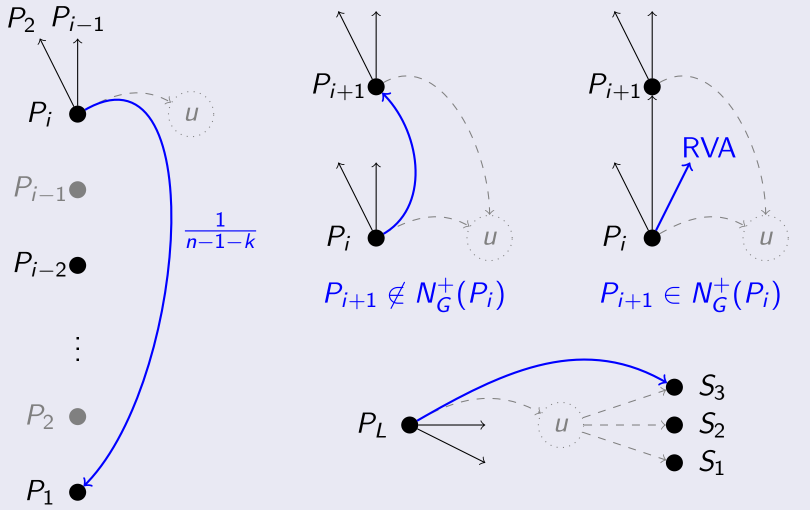


$L = |N_G^-(u)|$ and $1 \leq i \leq L-1$



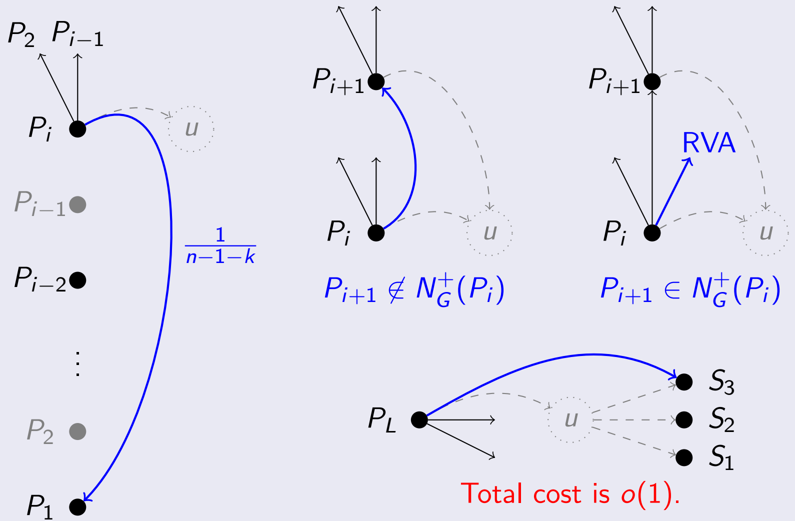
Deletion Algorithm: how to redirect loose edges ?

$L = |N_G^-(u)|$ and $1 \leq i \leq L-1$



Deletion Algorithm: how to redirect loose edges ?

$L = |N_G^-(u)|$ and $1 \leq i \leq L-1$



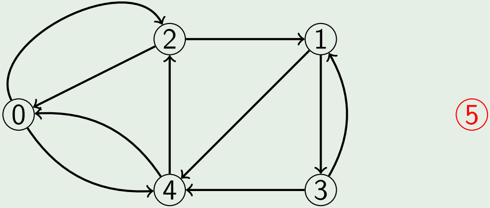
Total cost is $o(1)$.

Overview

- Introduction
- Update Algorithms
  - Deletion Algorithms
  - **Insertion Algorithms**
- Conclusion

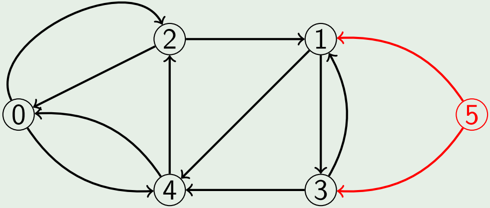
Insertion Algorithm: Simple insertion

Insertion of vertex 5



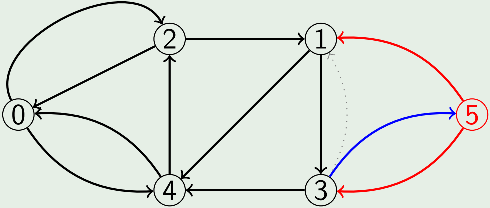
Insertion Algorithm: Simple insertion

Insertion of vertex 5



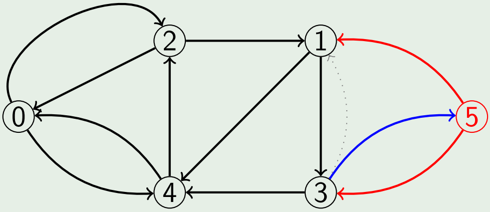
Insertion Algorithm: Simple insertion

Insertion of vertex 5



Insertion Algorithm: Simple insertion

Insertion of vertex 5



Simple Insertion Algorithm

- Draw k distinct vertices as successors, using `RV`
- Draw $X \sim \text{Binomial}(n, k/n)$, then pick X distinct random vertices as predecessors, and *steal* one edge from each of them
- Asymptotic cost: $k + k$

Insertion Algorithm: Using similar ideas as deletion

Best insertion algorithm

- Build first the predecessors of u by:
  - Choosing the number of predecessors using Binomial$(n, k/n)$
  - Starting from a call to `RV`
  - Using the *lost* vertex of the redirected edge to save some calls to `RV`
  - Last predecessor is used to produce a first successor for u
- Then choose the $k - 1$ other successors of u using `RV`

Cost and optimality

- Asymptotic Cost: k
- Optimal asymptotic cost, among bounded expected time algorithms

Overview

- Introduction
- Update Algorithms
  - Deletion Algorithms
  - Insertion Algorithms
- **Conclusion**

Conclusion

Distribution-preserving algorithms and k-out graphs

- Precise definition of distribution-preserving algorithms
- Several insertion and deletion algorithms for k-out graphs
- The most efficient algorithms are asymptotically optimal

Extension to more complex models

- Some *fixed* distribution for vertex out-degrees
- Undirected edges: e.g. regular graphs (difficult) or pairing models (easier)
- Geometric models: distribution of the network depends on some point set V

Connected works

- Possibility to maintain k-out graphs without knowing the size
- Expensive cost: difficult to simulate Binomial$(n, k/n)$

Thank you

- Introduction
- Update Algorithms
  - Deletion Algorithms
  - Insertion Algorithms
- Conclusion
- **Thank you for your attention**

Connected works

Without knowledge of the *size* ?

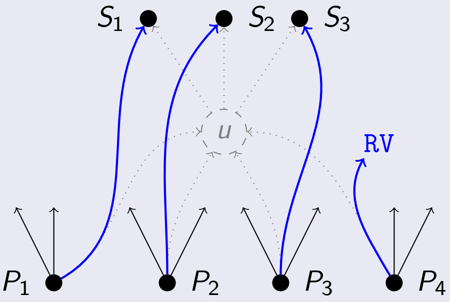
- Our algorithms need to simulate Bernoulli($a/(n-b)$) and Binomial($n, k/n$) using only `RV`
- Bernoulli($a/(n-b)$) can be simulated by one call to `RV`, using two sets $A, B \subset V$ such that $|A| = a$, $|B| = b$ and $A \cap B = \emptyset$
- With this simulation, all deletion/insertion algorithms presented here have the same cost: k for deletion, $2k +$ the cost of simulating the Binomial for insertion
- Binomial($n, k/n$) is known to be simulable but actually only in $\mathcal{O}(n)$ expected calls to `RV`

Special cases

- Binomial($n, 1/n$) can be simulated in 3.2 calls to `RV`

Deletion Algorithm: DEL2

DEL2 Schema

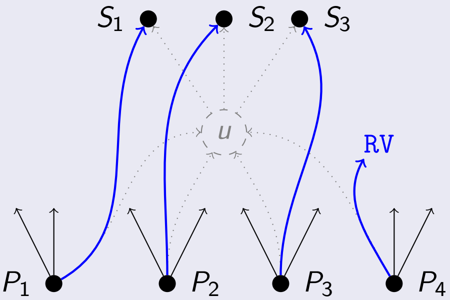


DEL2 Algorithm

- For $1 \leq i \leq k$, suggest to P_i:
  - With probability $\frac{i-1}{n-1}$, a uniform vertex of $\{P_1, \ldots, P_{i-1}\}$
  - With the remaining probability, S_i
- For any further predecessors and unsuccessful suggestions, use `RV` instead

Deletion Algorithm: DEL2

DEL2 Schema

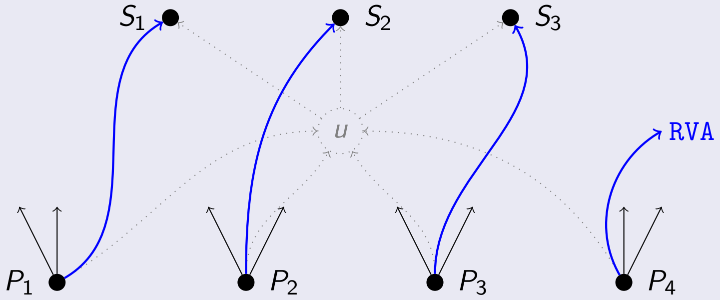


Cost

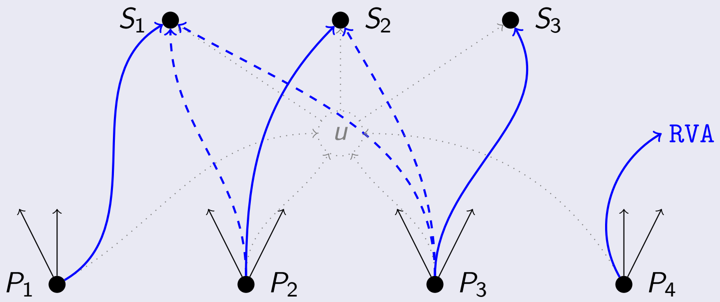
- Total asymptotic cost : $k \cdot \left(e^{-k} \cdot \frac{k^k}{k!} \right)$
- $\simeq \sqrt{\frac{k}{2\pi}}$

Deletion Algorithm: DEL2

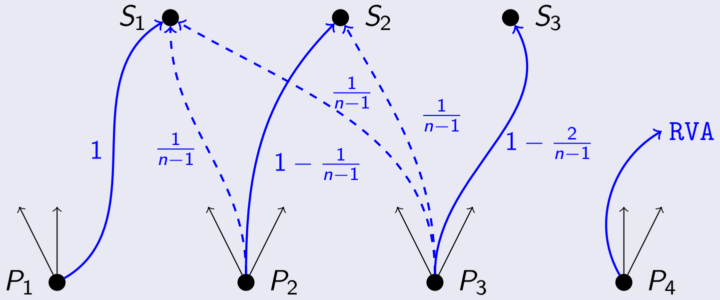
DEL2 Schema



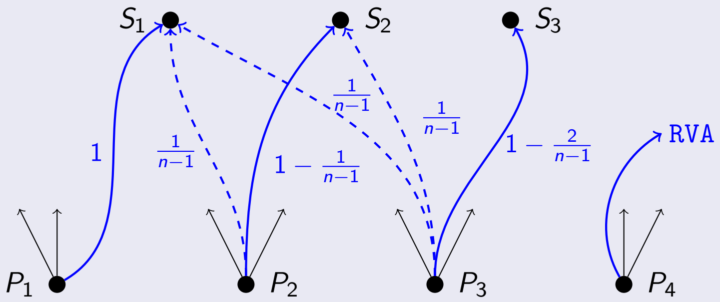
DEL2 Schema



DEL2 Schema


S_1
S_2
S_3
1
$\frac{1}{n-1}$
$1 - \frac{1}{n-1}$
$\frac{1}{n-1}$
$\frac{1}{n-1}$
$1 - \frac{2}{n-1}$
RVA
P_1
P_2
P_3
P_4

Del2 Schema



Cost

- Total asymptotic cost : $k \cdot \left(e^{-k} \cdot \frac{k^k}{k!}\right)$
- $\simeq \sqrt{\frac{k}{2\pi}}$

Deletion Algorithm: DEL3 algorithm

DEL3 Algorithm

- Replace u in P_i's neighbourhood by (in order):
  1. One of the $j \leq i-1$ *acceptable* and *already processed* predecessors, with probability $j/(n-1-k)$
  2. P_{i+1}, if the edge (P_i, P_{i+1}) does not already exist
  3. Some call to `RV` avoiding the j *acceptable* predecessors and P_i's successors, if (2) fails
- For the last predecessor, replace u by one of u's successors (then some call to `RV` if the suggestion is not accepted)

Deletion Algorithm: Del3 algorithm

Del3 Algorithm

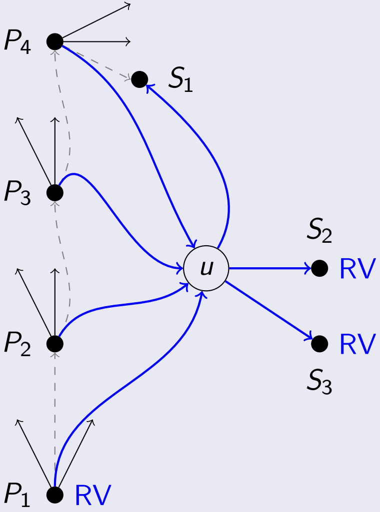
- Replace u in P_i's neighbourhood by (in order):
  1. One of the $j \leq i-1$ *acceptable* and *already processed* predecessors, with probability $j/(n-1-k)$
  2. P_{i+1}, if the edge (P_i, P_{i+1}) does not already exist
  3. Some call to `RV` avoiding the j *acceptable* predecessors and P_i's successors, if (2) fails
- For the last predecessor, replace u by one of u's successors (then some call to `RV` if the suggestion is not accepted)

Cost

- With probability $1 - \mathcal{O}(\frac{1}{n})$, we do not call `RV` at all
- Total cost is $o(1)$

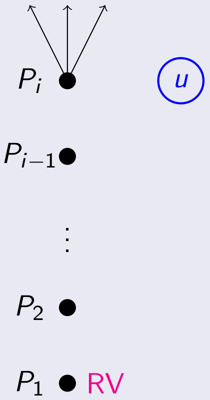
Insertion Algorithm: using similar ideas as DEL3

INS2 Schema



Insertion Algorithm: how to build $N^-_{G'}(u)$

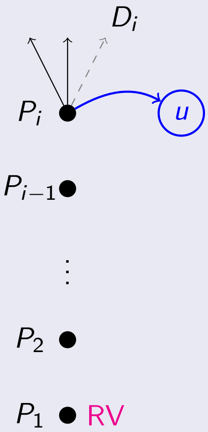
How to chose P_{i+1} ?



$L = |N^-_G(u)| \sim \text{Binomial}(n, k/n)$ and $2 \le i \le L$

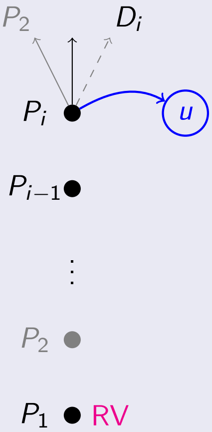
Insertion Algorithm: how to build $N^-_{G'}(u)$

How to chose P_{i+1} ?



$L = |N^-_G(u)| \sim \text{Binomial}(n, k/n)$ and $2 \leq i \leq L$

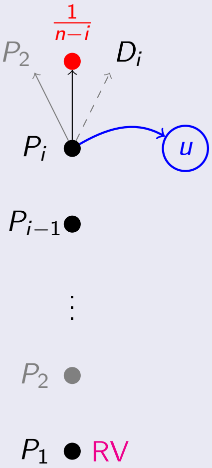
How to chose P_{i+1} ?



$L = |N^-_G(u)| \sim \mathrm{Binomial}(n, k/n)$ and $2 \leq i \leq L$

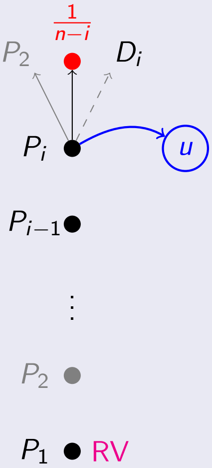
Insertion Algorithm: how to build $N^-_{G'}(u)$

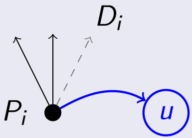
How to chose P_{i+1} ?



$L = |N^-_G(u)| \sim \text{Binomial}(n, k/n)$ and $2 \leq i \leq L$

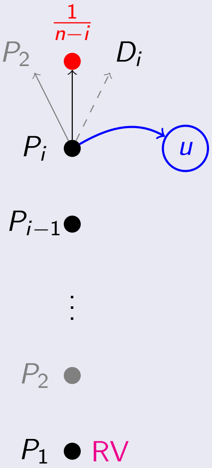
How to chose P_{i+1} ?

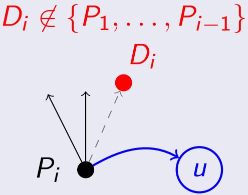




$L = |N^-_G(u)| \sim \text{Binomial}(n, k/n)$ and $2 \leq i \leq L$

How to chose P_{i+1} ?

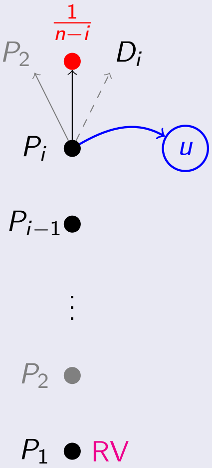


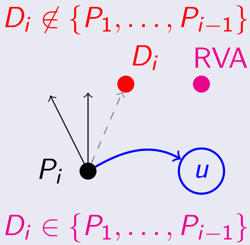


$L = |N^-_G(u)| \sim \text{Binomial}(n, k/n)$ and $2 \leq i \leq L$

Insertion Algorithm: how to build $N^-_{G'}(u)$

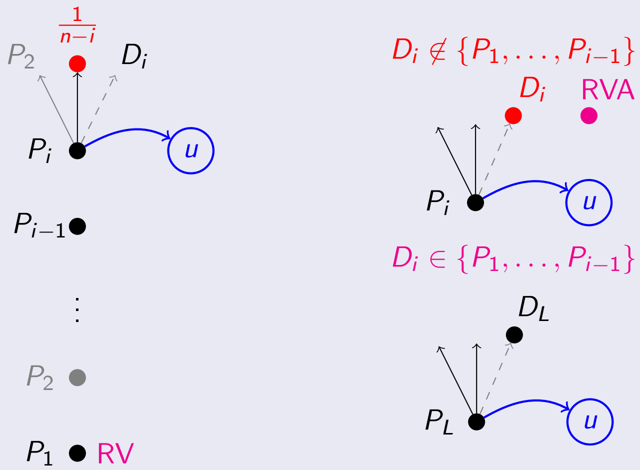
How to chose P_{i+1} ?





$L = |N^-_G(u)| \sim \text{Binomial}(n, k/n)$ and $2 \leq i \leq L$

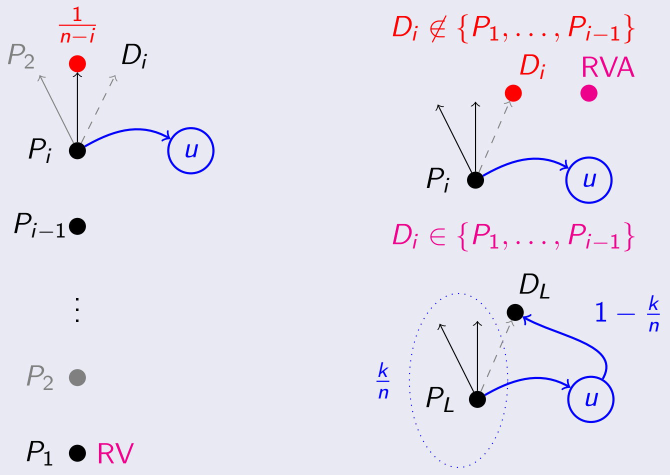
How to chose P_{i+1} ?



$L = |N^-_G(u)| \sim \text{Binomial}(n, k/n)$ and $2 \leq i \leq L$

Insertion Algorithm: how to build $N^-_{G'}(u)$

How to chose P_{i+1} ?



$L = |N^-_G(u)| \sim \text{Binomial}(n, k/n)$ and $2 \leq i \leq L$

Insertion Algorithm: Cost and optimality of INS2 algorithm

Cost of INS2 Algorithm

- One call to `RV` is needed to start the algorithm
- With probability $1 - \mathcal{O}(1/n)$, no calls to `RV` is needed in order to find P_{i+1} and $k-1$ calls are enough for the last step
- The asymptotic total cost is then $1 + (k-1)$

Insertion Algorithm: Cost and optimality of INS2 algorithm

Cost of INS2 Algorithm

- One call to `RV` is needed to start the algorithm
- With probability $1 - \mathcal{O}(1/n)$, no calls to `RV` is needed in order to find P_{i+1} and $k - 1$ calls are enough for the last step
- The asymptotic total cost is then $1 + (k - 1)$

Optimality (sketch) of INS2 Algorithm

- By counting possible graphs, we get that $k \log_2(n) + \mathcal{O}(1)$ new bits of information are needed to properly insert a vertex
- Each call to `RV` gives $\log_2(n)$ new bits of information
- Since other sources of randomness are available, one call to `RV` may be sufficient but results in a $\mathcal{O}(n)$ algorithm
- With asymptotic constant expected time complexity, k calls are needed

Insertion Algorithm: Ins2 algorithm

Ins2 Algorithm

- Choose $L \sim \text{Binomial}(n, k/n)$, then build a L-subset $P = P_1, P_2, \ldots P_L$ over V for u's predecessors
- P_1 is obtained through `RV`
- Then, for each $1 \leq i \leq L$: one of P_i's outgoing edge is chosen to be redirected to u, and we keep track of D_i the deleted destination, then P_{i+1} is obtained by (in order):
  1. One of the $j \leq k-1$ *acceptable* vertices among $N^+(P_i)$, with probability $j/(n-i)$
  2. D_i, if it is *acceptable*
  3. Some call to `RV` avoiding $P_1, \ldots, P_i$ and $N^+_G(P_i)$, if (2) fails
- Once all predecessors have been chosen, the first successor for u is obtained as follows:
  1. One vertex among $N^+(P_L) \setminus \{D_L\} \cup \{P_L\}$ is chosen uniformly with probability k/n
  2. "D_L" is used otherwise with the remaining probability
- Finally, the remaining $k-1$ successors are obtained using `RV`

Distributed Model of computation

Local features

- Deletion : need only to examine the underlying undirected graph at distance 2 from u
- Insertions : need to examine neighborhoods of vertices returned by `RV` or along short paths
- Possible implementation in a decentralized message-passing model
- Assumptions : Knowing a vertex *identity* is sufficient to contact it and we have access to the `RV` primitive

Out of the scope of this work

- Unreliable network
- Concurrency