Stochastic Graphs Have Short Memory: Fully Dynamic Connectivity in Poly-Log Expected Time

S. Nikoletseas(1)* **J. Reif**(2)†
P. Spirakis(1)*, **M. Yung**(3)

(1) Computer Technology Institute, Patras University, Greece
(2) Department of Computer Science, Duke University
(3) IBM Research Division, T.J. Watson Research Center

Abstract

This paper presents an *average case* analysis of *fully dynamic* graph connectivity (when the operations are edge insertions and deletions). To this end we introduce the model of *stochastic graph processes* (i.e. dynamically changing random graphs with random equiprobable edge insertions and deletions). As the process continues indefinitely, all potential edge locations (in $V \times V$) may be repeatedly inspected (and learned) by the algorithm. This learning of the structure seems to imply that traditional random graph analysis methods cannot be employed (since an observed edge is not a random event anymore). However, we show that a small (logarithmic) number of dynamic random updates are enough to allow our algorithm to re-examine edges as if they were *random with respect to certain events* (i.e. the graph "forgets" its structure). This *short memory* property of the stochastic graph process enables us to present an algorithm for graph connectivity which admits an *amortized expected* cost of $O(\log^3 n)$ time per update. In contrast, the best known deterministic worst-case algorithms for fully dynamic connectivity require $n^{1/2}$ time per update.

*This research was partially supported by the EEC ESPRIT Basic Research Projects ALCOM II and GEPPCOM

†Supported by DARPA/ISTO Contracts N00014-88-K-0458, DARPA N00014-91-J-1985, N00014-91-C-0114, NASA subcontract 550-63 of prime contract NAS5-30428, US-Israel Binational NSF Grant 88-00282/2, and NSF Grant NSF-IRI-91-00681.

1 Introduction

1.1 Dynamic Graph Problems and Previous Work

In dynamic graph problems one would like to answer queries on a graph that is undergoing a sequence of updates. A problem is called *fully dynamic* when the updates include both insertions and deletions of edges; this case is often more complex than the *partially dynamic* (where only one type of update is allowed, e.g. only insertions). The goal of a dynamic graph algorithm is to update the solution after a change (i.e. *on-line*), doing so more efficiently than recomputing it at that point from scratch. The adaptivity requirements usually make dynamic algorithms and dynamic data structures more difficult to design and analyze than their static counterparts.

Graph connectivity is one of the most basic problems with numerous applications and various algorithms in different settings. The best known result for worst-case fully-dynamic connectivity was (for quite a number of years) the very basic algorithm due to Frederikson ([10]) which takes $O(\sqrt{|E|})$ update time and which initiated a clustering technique. Very recently this was improved by a novel sparsification technique to $O(\sqrt{n}\log(|E|/n))$ by ([7]).

We remark that all the previous results ([10], [14], [11], [15] and [7]) on efficient fully-dynamic structures for general graphs were based on clustering techniques. This has led to solutions of an inherent time bound of $O(n^{\epsilon})$, for some $\epsilon < 1$, since the key problem encountered by these techniques is that the algorithm must somehow balance: (i) the work investing in maintaining the component of the cluster structure, and (ii) the work on the cluster structure (connecting the components).

1.2 Previous Work on Average Case Analysis of Algorithms

In this work we initiate the study of *average case* analysis of fully dynamic algorithms and techniques to achieve much more efficient (i.e., polylogarithmic per update) expected amortized time complexity. Designing algorithms with good average case behavior is directed towards better solving the typical case and better capturing (via probabilistic methods) its complexity– rather than analyzing the worst possible case. (A traditional caveat attached to this statement is that although the average case analysis gives an indication of a typical case, we have to be aware that, in fact, the actual space of certain practical problems may not be always known). For extensive surveys motivating the exploration of efficient average case algorithms and their probabilistic analysis see ([17]). Basic good average case graph algorithms were presented in various settings: sequential, parallel, distributed, NP-hard and so on (e.g. [2], [5], [18], [19], [20], [21], [23], [26], [29]). We are not aware of any investigation prior to ours concerning fully-dynamic graph theoretic problems.

1.3 Average Case Analysis of Dynamic Graph Algorithms

The investigation of the average case of fully dynamic graph suggests random graph updates. In this setting we would like to perform any sequence of three kinds of operations:

1. *Property-Query* (parameter): Returns *true* iff the property holds (or returns sub-graph as a witness to the property). For a connectivity query (u, v), a "true" answer means that the vertices u, v are in the same connected component.

2. *Insert* (x,y): Inserts a new edge joining x to y (assuming $\{x, y\}$ not in E).

3. *Delete* (x,y): Deletes the edge $\{x, y\}$ (assuming $\{x, y\} \in E$).

In this we assume random updates (insert and delete operations) have equal probability, 1/2. The edge to be deleted (inserted) is randomly chosen in E (E^c, i.e.,the set of edges not in E).

1.4 Our results

We consider the above randomly changing graphs, which thus give rise to a new probabilistic process on graphs which we call a *stochastic graph process* (which may be of independent interest). We then use this model to analyze our algorithm for connectivity.

The time complexity per update is $O(\log^3 n)$, both with high probability and on the average. Our complexity measure is amortized time (amortization over a long enough finite sequence of operations).

1.5 Our approach

We represent the graph as a spanning forest of rooted directed trees at some time periods (graph activation epochs) while at some other complementary time periods (the graph retirement epochs) we are forced to run a slow, deterministic algorithm. We show that the periods (i.e. sequence of updates) which require slow update time are *infrequent*. Our algorithm dynamically adapts to the dynamic changes of the stochastic object (the graph) by changing the data structure.

We choose to ignore graph edges until their inspection is indeed necessary. Non-inspected edges are kept in a *pool*. Pool edges, when used (inspected) for the first time, are proved to be *independent* of past history of the graph as far as certain events of the graph are concerned. However, during any update, a number of edges are inspected, an act which conditions their future use. Nevertheless, we show that the stochastic graph processes have an important property (which we call *short memory*) which allows our analysis to overcome these generated dependencies. Namely, after a period of a small (logarithmic) number of random updates, any edge inspected previously to that period can be re-used *as if it were random with respect to certain events*. This is so because the random updates change the graph and, hence, some properties seen at inspection may now *apply or not apply* independently of what had been observed about them before the period. We use the term *edge re-activation* to indicate the ability to re-use inspected edges after some number of updates. We feel that this property of the stochastic process, and algorithmic techniques like ours which exploit it, will play crucial roles in any average case analysis of fully dynamic graph algorithms.

2 Definition of Stochastic Graph Processes

In this section we define a process on the set of vertices of a graph.

Definition 1 *A stochastic graph process (sgp) on $V = \{1, 2, \ldots, n\}$ is a Markov Chain $G^* = \{G_t\}_0^\infty$ whose states are graphs on V.*

Definition 2 *A stochastic graph process on $V = \{1, 2, \ldots, n\}$ is called fair (fsgp) if*

1. *G_0 is the empty graph.*
2. *There is a $t_1 > 0$ such that $\forall t \leq t_1$, G_t is obtained by G_{t-1} by an addition of an edge uniformly at random among all edges in E_{t-1}^c. (Up to t_1 an edge is added at each $t \leq t_1$).*
3. *$\forall t \geq t_1$, G_t is obtained from G_{t-1} by either the addition of one edge, which happens with probability $1/2$ (all the new edges being equiprobable), or by the deletion of one existing edge, which happens with probability $1/2$ (and all existing edges are equiprobable to be selected).*

Remark: The stochastic graph process which includes only steps (1) and (2) above and $t_1 \leq \binom{n}{2}$ was used by Erdös and Renyi ([8]) to define $G_{n,M}$, which was called the *random graph process* as the number of inserted edges M progresses.

Next notice that:

Lemma 1 *Let G^* be an fsgp on $V = \{1, 2, \ldots, n\}$. Let $G_t = (V, E_t)$ be the state of the process at time t, then G_t is also a random graph from $G_{n,M}$ (for some M).*

The fsgp will be used as the model of our dynamic algorithm. The "steps" denoted by the variable t (time) are those which change the graph, namely the insertions and deletions.

We may assume (wlog) that we start from a random G_{t_1}, which is approximately a graph in $G_{n,p}$ of $p = c/n$ (for appropriate large enough c) where $t_1 = p\binom{n}{2}$, (see ([3]) for results on the close approximation between the random graph spaces with respect to various properties). The initial stage (of building by insertions only) can be easily handled (e.g., by a union-find algorithm).

We may investigate the behavior of stochastic graphs when the edge set cardinality is bounded from above (or possibly from below as well) by a bound on the number of allowed edges. This allows us to deal with families of graphs of interest which are not uniform. An important example is the relatively sparse graphs between $O(n)$ and $O(n \log n)$ edges (which are used as an example in ([19])), which can be captured by the following constraints:

Definition 3
1. *An fsgp is called truncated from below by L (denoted $tb - fsgp(L)$) if the edge deletion probability becomes zero (rather than $1/2$) for each $t > t_1$ such that $|E_t| \leq L$ (where $0 \leq t_1 \leq L \leq \binom{n}{2}$ is called the mininum number of edges).*
2. *An fsgp is called truncated from above by U (denoted $ta - fsgp(U)$) if the edge insertion probability becomes zero (rather than $1/2$) for each $t > t_1$ such that $|E_t| \leq U$ (where $0 \leq t_1 \leq U \leq \binom{n}{2}$ is called the maximum number of edges).*
3. *An fsgp is called bounded by L, U (denoted $b - fsgp(L, U)$) if it is both $ta - fsgp(U)$ and $tb - fsgp(L)$ for $L \leq U$.*

3 General Description of the Algorithm

3.1 The Input

We assume the input to the algorithm to be a stochastic graph process of the form b-fsgp(L,U) with $L = cn$ (for an appropriate constant c greater than the giant component threshold for random graphs) and $U = \binom{n}{2}$. We can also handle sparser graphs (applying, w.l.o.g., deterministic maintenance to this range); in fact, the adaptation of the graph and treating it in various graph epochs according to the current conditions is a first technique which is employed. For purpose of the analysis we will first assume graphs with $U = \lambda n \log n$ (where λ is below the connectivity threshold constant for random graphs). Denser graphs results easily follow.

3.2 The Data Structures and the Edge Re-Activation Technique

Our algorithm maintains a forest of spanning trees, one per connected component of the graph. The algorithm guarantees that the tree $T(\Gamma)$ of the graph's giant component Γ, has diameter bounded above by $\Delta = \lambda_1 \log^2 n$ most of the time, with high probability ($\lambda_1 > 1$ is an appropriate constant). The trees are *rooted* and the roots are suitably maintained. The tree is directed towards the root; each tree node points to a neighbor in the direction of the root. The pointer directions are maintained within the stated time bounds. (Where the context is clear, we simply denote $T(\Gamma)$ by T).

In addition to the forest of rooted trees, we also maintain the graph adjacency matrix and each edge in the matrix may have a status label which is one of (1) "tree" edge (2) "retired" edge or (3) "pool" edge.

Active edges are pool or tree edges. The pool can be imagined to be a set of unused random edges, useful for our dynamic maintenance operations. The pool is being exhausted by edges selected to be used in insert/delete operations. We look at many edges at one operation and discard them while at most we add one edge to the pool. Thus, we may soon exhaust it and have no "pool", as we have looked at the entire graph and cannot use the fact that it is random. This seems like an inherent problem. However, we develop a technique which allows us to refill the pool dynamically with edges that could be seen as "random" for later operations, in two ways: (1) from superfluous insertions and (2) from the set of retired edges, because (as we shall show later) retired edges are *reactivated* after a certain number of random insertions and deletions counted from the time they became retired.

Edge reactivation allows re-use of seen edges efficiently, because their past use conditions only a certain number of future operations and then this bias is "forgotten" by the stochastic process, due to the effect of the random updates in the time passed.

We keep for each vertex its component name (e.g. the root of the corresponding tree). This component name allows connectivity queries to be answered in $O(1)$ time with certainty. For each tree we also keep its size (in number of vertices).

Our algorithm's main goal is, then, to maintain the correct labeling of the connected components.

3.3 The Dynamic Algorithm

3.3.1 High Level Description: Graph Activation and Graph Retirement Periods

The algorithm is initialized by running the linear expected time connected components of ([19]) by using a fraction of at least cn edges of the graph (c has to be selected much larger than the constant of that algorithm). This construction guarantees the creation of a random spanning tree in each component. With probability at least $1 - n^{-\alpha_1}$ (where $\alpha_1 > 2$ a constant) the algorithm constructs a giant component (call it Γ) and many small ones (the giant component has at least ϵn vertices and the sizes of the other components are at most $O(\log n)$)(see e.g. ([3])).

With small probability ($\leq n^{-\alpha_1}$) the construction fails to produce a giant component. Then, we enter a *graph retirement epoch.* The giant component tree construction is called a **total reconstruction**. A *successful* total reconstruction produces a giant component and logarithmic ones, with a $\log n$-diameter tree spanning the giant component. The algorithm then enters a *graph activation epoch.*

The graph process proceeds forever through a sequence of graph epochs which alternate between graph activation and graph retirement epochs.

A graph activation epoch is enterd by a *successful* total reconstruction. It lasts at most A operations (A is equal to $\lambda_1 n^2 \log n$ where $\lambda_1 > 1$ is a constant); after A operation a reconstruction is attempted. The epoch may (with very small probability) end before all A operations are done. This can happen only when a deletion operation disconnects the giant component tree and the attempted fast reconnection of the tree (by the excess existing edges) fails.

A graph retirement epoch starts when the (previous) graph activation epoch ends. It lasts at least R operations (R is equal to $\lambda_2 n \log^2 n$, where λ_2 is a constant > 1). At the end of a graph retirement's R operations, a total reconstruction is attempted. As we show, it will succeed with high probability. In such a case, a new graph activation epoch is entered. However, with small probability, the total reconstruction may fail. Then, the graph retirement epoch continues for another set of R operations and again a total reconstruction is attempted. This may continue until a successful total reconstruction.

The formal structure of the dynamic algorithm is presented in Appendix 1.

3.3.2 The Dynamic Operations in a Graph Activation Epoch

In a graph activation epoch we use the data structures to process updates at $\log^3 n$-time cost per operation with high likelihood, as we will show in the analysis section.

Inductively assume during an activation period that there are $\Lambda_1, \epsilon, \alpha$ ($0 < \epsilon < 1, \Lambda_1 > 0, \alpha > 1$):
(a) The distance between the root and any node in the giant tree T cannot exceed $\Lambda_1 \log^2 n$,
(b) At least a constant fraction $\epsilon|T|$ of the giant tree nodes are at distance up to $\alpha \log n$ from the root (e.g., $\epsilon \geq 1/2$).

The induction holds at the beginning of an activation period (basis), by construction. Assume that it holds before a random edge update.

Definition 4 *Let a* place *be a node of the giant tree satisfying (b) of the induction hypothesis.*

We perform the insertion-deletion operations according to the following rules:

- **Insertion of a random edge:**

  Case b1: The edge joins vertices of the same tree. We just put it in the pool.

  Case b2: The new edge joins two trees. We update the root and the component name by changing the name and pointers at nodes of the smaller tree. (The time needed is linear in the size of the smaller tree).

- **Deletion of a random edge**

  Case c1: The edge is either a pool or a retired edge. We just delete it from all data structures.

  Case c2: The edge is a tree edge of a small tree. We sequentially update the small component (it will either reconnect by a pool edge or it splits into two small trees, in which case we relabel the smaller of the two). This also takes linear time in the tree size.

  Case c3: The edge is a tree edge of the tree T of the giant component. Let the deleted edge be $e = \{u, v\}$. Let $T(u), T(v)$ be the two pieces in which the tree is split by the removal of e, such that $u \in T(u)$ and $v \in T(v)$. From each of the vertices u, v we start a procedure called neighborhood search. Each neighborhood search consists of a sequence of phases. The two neighborhood searches out of u, v are interleaved in the sense that we execute one phase of each in turn. Here we present the neighborhood search out of u: ($k_1 > 1$ is an appropriate constant).

Neighborhood Search (u):

Start a breadth-first-search (BFS) out of u until $k_1 \log n$ nodes are visited. The visit of the ith node designates the start of the ith phase. A phase may return “success” or “failure”.

The neighborhood search may:

1. Finish with no success, in which case the algorithm undergoes total reconstruction and the current activation epoch is ended (this may be called a “fatal event”).
2. The nodes connected to u may be exhausted before the search finishes. In this case the giant component is just disconnected into a still-giant and a $O(\log n)$ piece. In this case, we just rename the midget component.
3. The search may report a number of successes (successful reconnection of the two tree pieces). We then choose the reconnection which is closer to the root and reconnect the two pieces.

Phase i (visit of node w): If w has no pool edges out of it then end this phase. Else, choose one pool edge out of w at random, out of its pool edges. Let this edge be $e' = \{w, w'\}$.

1. Check whether u points to v by its pointer towards the root R of the tree, or vice-versa. If u points to v then R belongs to $T(v)$ else R belongs to $T(u)$. Since w is in the same piece with u, we follow the path from w' to the root R (inductively this takes $O(\log^2 n)$ time). If e is not used in that path and $R \in T(v)$ then e' reconnects the two pieces. In all other cases, e' does not reconnect the two pieces. In such a case, end the phase and return “failure” after putting e' in retirement.
2. Provided that $e' = \{w, w'\}$ reconnects the two pieces, assume without loss of generality that $R \in T(v)$. We follow the path from u to the root and in $O(\log^2 n)$ time we determine the distance $d(u, R)$ of u from the root in the tree before the reconnection. Let $d(u, w)$ be the distance from u to w in $T(u)$. Check whether $d(u, R) \geq (k_1 + \alpha) \log n$ (i.e. whether the piece chopped away is a distant one). Also check whether $d(w', R) < \alpha \log n$ (Test (*)), where $d(w', R)$ is the distance of w' from R in $T(v)$.

In the analysis we show that, for each e' which reconnects the two pieces, test (*) is satisfied with probability at least 1/2. Since we try $\Theta(\log n)$ such e', test (*) will be found to hold at least once with very high probability. Note that (*) implies that, in the reconnected new tree,

$$d_{new}(u, R) \leq (k_1 + \alpha) \log n$$

The search returns “success” in the following two cases:
Case 1: $d(u, R) \leq (k_1 + \alpha) \log n$ and test (*) holds (Type-1 reconnections).
Case 2: $d(u, R) > (k_1 + \alpha) \log n$ and test (*) holds (Type-2 reconnections).
Note that Type-2 reconnections never increase the giant tree’s diameter. However, Type-1 reconnections may increase the giant tree’s diameter by at most $k_1 \log n$.
The idea here is that, when a random deletion chops off a “distant” piece from the tree then we succeed whp to reconnect it *closer* to the root (Type-2 reconnections). The reconnections of pieces that were already “close” may increase the diameter at most logarithmically, but we will show that the cumulative effect of such increments *will respect the induction hypothesis* over the whole length of the graph activation period.

(end of phase i)
(end of neighborhood search)

If the neighborhood search ends up with success then the giant tree is reconnected (by the successful e') and the graph activation epoch continues.

3.3.3 Edge Reactivation

Graph activation epochs are partitioned into *edge deletion intervals* of $k_3 \log n$ deletions (k_3 an appropriate constant). All edges put in retirement during an edge deletion interval are returned into the pool after a delay of another edge deletion interval. We do not reactivate edges of small components (midgets). Of course each edge under retirement is actually reactivated at a different time (so edge retirement periods do not correspond necessarily to edge deletion intervals) but we "bulk" edges to be reactivated in blocks for reasons of algorithmic analysis. (see section 4.2)

Also, after a total reconstruction, all edges not seen by the connected components algorithm are put in the pool.

4 The Analysis of the Algorithm

4.1 Analysis of the Graph Activation Epochs

4.1.1 The giant tree reconnects successfully with high probability

Consider two (coupled) phases (i) of the neighborhood searches out of u and v in a random deletion of a giant tree edge e. Each edge $\{w, w'\}$ tried in a phase, is taken from the pool. Hence, it is a non-tree edge which is random with respect to its other end w' (to prove that, see the analysis of edge reactivation). If A, B are the pieces in which the tree T has been broken, let w.l.o.g. $|A| \geq |B|$ and consider in the two coupled phases the edge $\{w, w'\}$ emanating from B. Condition on its existence. Then

$$\Pr\{\{w, w'\} \text{ is such that } w' \in A | \{w, w'\} \text{ exists in pool}\} = \frac{|A|}{|A| + |B|} \geq \frac{1}{2} \tag{1}$$

Total reconstruction, if successful, provides a *random graph* of at least cn edges (for a proof of this see section 4.3). Here c is selected to be a constant much greater then α' ($\alpha' n$ are the edges used by the Karp-Tarjan algorithm during total reconstruction). Thus, the average number of edges emanating from each vertex and belonging in the pool is initially (i.e. in the beginning of the activation phase) $c - \alpha'$.

Lemma 2 *For each node w visited in Neighborhood Search:*

$$\Pr\{\textit{at least one pool edge out of } w \textit{ exists}\} \geq \frac{1}{4}$$

Proof : See Appendix 2 (section 6.1).

Let E_1 = the event that the edge $e = \{w, w'\}$ indeed reconnects the tree T, given $\{w, w'\}$ exists. Let E_2 = the event that $\{w, w'\}$ returns success in the neighborhood search, given E_1. From (1), $\Pr\{E_1\} \geq 1/2$.

Lemma 3 $\Pr\{E_2 | E_1\} \geq \frac{1}{4}$

Proof: See Appendix 2 (section 6.2).
From the above it follows that:

Lemma 4 *Each neighborhood search succeeds with probability at least* $1-n^{-\gamma}$. *(γ a constant which can be made as large as we want by increasing k_1).*

Proof: See Appendix 2 (section 6.3).
Next we estimate the expected (and with high probability) number of successes in each neighborhood search.

Lemma 5 *The number of successes in each neighborhood search is* $\Theta(\log n)$ *with high probability.*

Proof: See Appendix 2 (section 6.4).

Theorem 1 *The induction hypothesis is preserved for the whole length* $A = \lambda_1 n^2 \log n$ *of the graph activation period, with probability at least* $1 - n^{-\delta}$, $\delta > 0$ *a constant.*

Proof: Consider the sequence S of insertions and deletions (of length A) of the graph activation period. Since they are random and equiprobable, by Chernoff bounds, for any small $\beta \in (0,1)$ we have:

$$\Pr\left\{\text{number of insertions of } S \leq (1+\beta)\frac{\lambda_1}{2} n^2 \log n\right\} \geq 1 - \exp\left(-\frac{\beta^2}{2}\frac{\lambda_1}{2} n^2 \log n\right)$$

and

$$\Pr\left\{\text{number of deletions of } S \leq (1+\beta)\frac{\lambda_1}{2} n^2 \log n\right\} \geq 1 - \exp\left(-\frac{\beta^2}{2}\frac{\lambda_1}{2} n^2 \log n\right)$$

By Lemma 4, the probability that the giant tree is fatally disconnected during the same operation in S is:

$$\begin{aligned} Q &= \Pr\{\exists \text{deletion in which neighborhood fails}\} \leq \sum_{\text{all deletions}} (\text{neighborhood search fails}) \leq \\ &\leq n^2 \log n \, n^{-\gamma} \leq n^{-(\gamma-3)} \end{aligned}$$

Thus:

$$\Pr\{\text{Giant tree is not fatally disconnected during the whole } S\} \geq 1 - n^{-(\gamma-3)}$$

Let V_1 be the event "the giant tree is not fatally disconnected for the whole sequence S". Condition S on V_1.
We first argue about insertions: Since the number of non-giant components is less than n, all such components will be joined to the giant tree by at most n insertions each of which joins a midget with a giant tree. Each such insertion adds at most an increment of $O(\log n)$ to the giant tree's diameter. Call these insertions "*incremental insertions*". The giant tree initially has ϵn "places". Since insertions are random, the probability that an incremental insertion hits a particular place P is $\frac{1}{\epsilon n}$. But the number of nodes not in the giant tree is $\Theta(n)$, the conditional probability p_c that the inserted edge will hit a small component, given that it starts from a particular place, is $\frac{\Theta(n)}{\begin{pmatrix} n \\ 2 \end{pmatrix} - |E|}$ because the possible insertions are $\begin{pmatrix} n \\ 2 \end{pmatrix} - |E|$. Thus p_c is $\Theta(\frac{1}{n})$. Let $\sigma_1 > 0$ be such that $p_c = \frac{\sigma_1}{n}$.
By Chernoff bounds then, $\forall \beta \in (0,1)$, $\sigma_2 > 1$ we have:

$$\Pr\left\{\text{number of incremental insertions at } P > (1+\beta)\left(\frac{1}{\epsilon n}\frac{\sigma_1}{n}\right)\sigma_2 n^2 \log n\right\} \leq$$
$$\leq \exp\left(-\frac{\beta^2}{2\epsilon}\sigma_1\sigma_2 \log n\right) = n^{-\frac{\beta^2\sigma_1\sigma_2}{2\epsilon}}$$

Let $\gamma_1 = \frac{\beta^2\sigma_1\sigma_2}{2\epsilon}$. By controlling σ_2, γ_1 can be made as large as we want. Let $\sigma = \sigma_1\sigma_2$. Then:

$$\Pr\left\{\text{number of incremental insertions at a place } P \; > \sigma\frac{1+\beta}{\epsilon}\log n\right\} \leq n^{-\gamma_1} \tag{2}$$

Definition 5 *Let Δ_I be the total giant tree diameter increase in S due to insertions only. Let Δ_D be the total giant tree diameter increase in S due to deletions only.*

By (2), since $\Delta_I \leq \log n\times$ (max number of insertions which are incremental at the same place P), we get:

$$\Pr\left\{\Delta_I \geq \frac{\sigma(1+\beta)}{\epsilon}\log^2 n\right\} \leq n^{-\gamma_1} \tag{3}$$

By similar arguments (see Appendix 2: section 6.5) we prove that:

$$\Pr\left\{\Delta_D \geq \sigma'(1+\beta')(1+\beta)\frac{\lambda_1}{2\epsilon}\log^2 n\right\} \leq n^{-\gamma_2}$$

Since the total diameter increase in S is bounded above by $\Delta_I + \Delta_D$, we get for $\gamma_3 = \max\{\gamma_1, \gamma_2\}$, that:

$$\Pr\{\text{Total diameter increase in } S \text{ is } \Theta(\log^2 n) \text{ given event } V_1\} \geq 1 - n^{-\gamma_3}$$

But $\Pr\{V_1\} \geq 1 - n^{-(\gamma-3)}$. Let $\gamma_4 = \max\{\gamma_3, \gamma - 3\}$. Hence

$$\Pr\{\text{ The diameter of the giant tree is } \Theta(\log^2 n) \text{ in whole } S\} \geq 1 - n^{-\gamma_4}$$

which proves (a) of the induction hypothesis, for

$$\Lambda_1 = \frac{\sigma(1+\beta)}{\epsilon} + \frac{\sigma'(1+\beta')(1+\beta)\lambda_1}{2\epsilon}$$

Since insertions and deletions are at random places, part (b) is preserved also. □

Corollary 1 *During the whole activation period, the time for an operation is $\Theta(\log^3 n)$ with probability $\geq 1 - n^{-\gamma_4}$.*

Proof: See Appendix 2 (section 6.6).

Theorem 2 *Conditioning on the fact that the graph undergoes a graph activation epoch and provided that the pool is reactivated successfully, each operation in the epoch takes at most $O(\log^3 n)$ time, with probability $\geq 1 - n^{-\delta}$.*

Proof: See Appendix 2 (section 6.7).

4.2 Edge Reactivation with High Probability

Definition 6 *Let x be an edge chosen from the pool and unsuccessfully used to reconnect the two pieces A, B of the tree T which corresponds to the giant component, created by the deletion D of an edge e. Let $L(D)$ be the shortest possible sequence of insertions/deletions following D, such that the last operation in $L(D)$ is a deletion, breaking the tree into pieces A', B' where, by again using x, the probability of reconnecting A', B' is at least as it was when x was used to reconnect A, B. Then $|L(D)|$ is called the retirement interval of edge x. (i.e. we want the information that a failed to certifiably reconnect A, B to be "lost"). Then we say that x has been reactivated and can be put in the pool of "random" edges.*

Lemma 6 *A failed edge x (as above in Def. 6) will be reactivated when:*

1. *An insertion or a tried edge from pool reconnects the tree (and A, B) and*
2. *The first of the subsequent deletions happens with the property that it deletes a random edge from the tree T.*

Proof: Let $x = \{u, v\}$. Let I_1 be the event "after the deletion of a random edge e, x will reconnect the two pieces A, B, for the first time of Def. 6" and I_2 be the event "after A, B are again reconnected by a tried pool edge and a random edge e' is deleted for the first time, x will reconnect A, B ". We must show:

$$\Pr\{I_1\} \leq \Pr\{I_2/\bar{I}_1\}$$

Let *path*(u, v) be the path connecting u, v on the tree T before the deletion of e. Then $\Pr\{I_1\} = \Pr\{e \in \mathit{path}(u, v)$ and either u or v are close to the root of $T\}$.

Now, consider $I_2/\bar{I}_1$. For x to have been *tried* and *failed*, it must have been the case that (1) x was at the neighborhood of e and (2) (a) either both u, v belonged to the same piece of T or (2) (b) both were away from the root of T, even when they belonged to different pieces. Notice that the distance of u, v from the root of T *does not change in each piece* from the time x is tried to the time x is re-tried.

Since x is tried again, it belongs to the neighborhood of e'. Thus the neighborhoods of e and e' *intersect.* This means that e' belongs to the part of tree T *which is unchanged up to the deletion of* e' (new insertions of tree edges are happening at the leaves.). Thus, for x to succeed, it again must be the case that $e' \in \mathit{path}(u, v)$ in the new tree, with the relevant possible choices of e' the same as for e' or better. Thus $\Pr\{I_1\} \leq \Pr\{I_2/\bar{I}_1\}$.

Definition 7 *In Def. 6, if x reconnected A, B but failed the diameter test, we call it a positive edge. If x just connected A to itself we call x a negative$-A$ edge. If x connected B to itself, we call x a negative$-B$ edge.*

Lemma 7 *For each edge x which was used and failed (except for midget edges) there exist $\lambda_1 > 2$, $\lambda_2 > 2$ such that with probability $\geq 1 - n^{-\lambda_2}$*

$$\mathit{retirement} - \mathit{interval}(x) \leq \lambda_1 \log n$$

where $\mathit{retirement} - \mathit{interval}(x)$ is the length of the retirement interval for edge x.

Proof: See Appendix 2 (section 6.8).

4.3 Analysis of the Graph Retirement Epochs

The purpose of the graph retirement epoch is to re-activate the set E_1 of edges in the giant tree (which may not be so big) and the smaller trees (the pool edges are "fresh"). Since there are at most $c_1 n$ such edges (c_1 a constant), a graph retirement epoch needs to delete the whole set E_1. Each random deletion hits (deletes) an edge in E_1 with probability $\Theta(\frac{1}{n})$. Thus,

$$\Pr\{\text{a particular } e \in E_1 \text{ avoids to be deleted for } \lambda_2 n \log n \text{ operations}\} \leq$$
$$\leq \left(1 - \frac{1}{n}\right)^{\lambda_2 n \log n} \leq e^{-\lambda_2 \log n} = n^{-\lambda_2}$$

Thus

$$\Pr\{\text{all } E_1 \text{ is deleted in graph retirement}\} \geq 1 - n^{-(\lambda_2 - 1)}$$

By Chernoff bounds, with probability $\geq 1 - n^{-\delta_1}$ ($\delta_1 > 1$) the number of insertions and deletions in the retirement period are of the same order. Also, with probability 1/2 (due to randomness) the number of insertions exceeds the number of deletions. Thus, at the end of each $\lambda_2 n \log n$ operations (call each such number of consecutive operations a *run* of the graph retirement period) we have an instance of a random graph again of cn edges (by Lemma 1 and the fact that the worst case algorithms used to serve the operations *do not condition any events* in the instance). Thus, a total reconstruction at the end of the run is successful with probability at least

$$\frac{1}{2}(1 - n^{-\delta_1})(1 - n^{-\alpha_1}) \geq \frac{1}{3}$$

(where α_1 is the constant in Karp-Tarjan ([19]) algorithm). Hence, the number of total reconstructions y before successfully entering a graph activation period will be

$$\Pr\{y \geq k \log n\} \leq \left(1 - \frac{1}{3}\right)^{k \log n} = n^{-k \log 3}$$

Thus, if $k_1 = k \log 3$

Lemma 8 *With probability $\geq 1 - n^{-k_1}$ the number of runs in the graph retirement period is at most $k \log n$ and thus its total length is at most $k \lambda_2 n \log^2 n$.*

4.4 The Amortized Expected Time Analysis of the Algorithm

Important Remark: Each reconnection (after an edge deletion) of the giant tree has expected time $O(\log^3 n)$ with high probability. In order to speak about amortized expected time, this has to be done for a finite sequence of operations (else, the linearity of expectation will not apply to infinite sequences and we may need the bounded convergence theorem). It follows that:

Theorem 3 *Our algorithm has $O(\log^3 n)$ amortized cost per operation, with high probability (or amortized expected) in the sense that a long sequence of M operations takes expected time $O(M \log^3 n)$, for large enough M.*

Proof: See Appendix (section 6.9).

References

[1] A. Aho, J. Hopcroft and J. Ullman, “The Design and Analysis of Computer Algorithms”, Addison-Wesley, Reading, MA, 1974.

[2] D. Angluin and L. Valiant, “Fast Probabilistic Algorithms for Hamiltonian Circuits and Matchings”, JCSS, vol. 18, pp. 155–193, 1979.

[3] B. Bollobás, “Random Graphs”, Academic Press, 1985.

[4] J. Cheriyan and R. Thurimella, “Algorithms for parallel k-vertex connectivity and sparse certificates”, 23rd STOC , pp. 391–401, 1991.

[5] D. Coppersmith, P. Raghavan and M. Tompa, “Parallel graph algorithms that are efficient on the average”, 28th FOCS, pp. 260–270, 1987.

[6] D. Eppstein, G. Italiano, R. Tamassia, R. Tarjan, J. Westbrook and M. Yung, “Maintenance of a minimum spanning forest in a dynamic plane graph”, 1st SODA, pp. 1–11, 1990.

[7] D. Eppstein, Z. Galil, G. Italiano and A. Nissenzweig, “Sparsification- A technique for speeding up Dynamic Graph Algorithms”, FOCS 1992.

[8] P. Erdös and A. Renyi, “On the evolution of random graphs”, Magyar Tud. Akad. Math. Kut. Int. Kozl. 5, pp. 17–61, 1960.

[9] W. Feller, “An Introduction to Probability Theory and its Applications”, John Wiley, New York, 1957.

[10] G. Frederikson, “Data structures for on-line updating of minimum spanning trees”, SIAM J. Comput. , 14, pp. 781–798, 1985.

[11] G. Frederikson, “Ambivalent data structures for dynamic 2-edge-connectivity and k-smallest spanning trees”, 32nd FOCS, pp. 632–641, 1991.

[12] G. Frederikson, “A data structure for dynamically maintaining rooted trees”, 4th SODA, 1993.

[13] A. Frieze, Probabilistic Analysis of Graph Algorithms, Feb. 1989.

[14] Z. Galil and G. Italiano, “Fully dynamic algorithms for edge connectivity problems”, 23rd STOC, 1991.

[15] Z. Galil, G. Italiano and N. Sarnak, “Fully Dynamic Planarity Testing”, 24th STOC, 1992.

[16] R. Karp, “Probabilistic Recurrence Relations”, 23rd STOC, pp. 190–197, 1991.

[17] R. Karp, J. Lenstra, C. McDiarmid and A. Rinnoou Kan, “Probabilistic analysis of combinatorial algorithms: an annotated bibliography”, in Combinatorial Optimization: annotated bibliography, ed. N. O’Heigeartaigh, J. Lenstra and A. Rinnoou Kan, John Wiley, New York, 1984.

[18] R. Karp and M. Sipser, “Maximum matching in sparse random graphs”, 22nd FOCS, pp. 364–375, 1981.

[19] R. Karp and R. Tarjan, “Linear expected time for connectivity problems”, 12th STOC, 1980.

[20] L. Kučera, “Canonical labeling of regular graphs in linear expected time”, 28th FOCS, pp. 271–279, 1987.

[21] R. Motwani, “Expanding graphs and the average-case analysis of algorithms for matching and related problems”, 21st STOC, pp. 550–561, 1989.

[22] A. Moffat and T. Takaoka, “An all pairs shortest path algorithm with expected running time $O(n^2 \log n)$”, 26th FOCS, pp. 101–105, 1985.

[23] S. Nikoletseas and P. Spirakis, “Near-Optimal Dominating Sets in Dense Random Graphs in Polynomial Expected Time”, 19th International Workshop on Graph-Theoretic Concepts in Computer Science (WG), 1993.

[24] J. Reif, “A topological approach to dynamic graph connectivity”, Inform. Process. Lett. , 25, pp. 65–70, 1987.

[25] J. Reif and P. Spirakis, “Expected parallel time analysis and sequential space complexity of graph and digraph problems”, Algorithmica, 1992.

[26] E. Shamir and E. Upfal, “N-processors graphs distributively achieve perfect matching in $O(log^2 n)$ beats” ACM PODC pp. 238–242.

[27] D. Sleator and R. Tarjan, “A data structure for dynamic trees”, J. Comput. System Sci., 24, pp. 362–381, 1983.

[28] J. Spencer, “Ten Lectures on the Probabilistic Method”, SIAM, 1987.

[29] P. Spira and A. Pan, “On finding and updating spanning trees and shortest paths”, SIAM J. Comput., 4, pp. 375–380, 1975.

5 APPENDIX 1: The Algorithm (high level description)

```
Input: A stochastic graph process starting from $G_{n,p}$ with $p = \frac{c}{n}$ at the beginning.
begin (Algorithm)
initialize: Run Connected Components of ([19]) on the input to construct a giant component
(with a spanning tree of logarithmic diameter). If the construction
fails then go to C else go to B.
do forever
begin (main loop)
B: (Graph Activation Epoch)
  for $A = \lambda_1 n^2 \log n$ insertions and deletions do
  begin
    case of
          random insertion: Perform it (cases b1, b2 of section 3.4.2)
          random deletion: Perform it (cases c1, c2, c3 of section 3.4.2). If it breaks
          the giant tree, then attempt to reconnect the giant tree (as
          explained in "Deletion of a Random Edge" below).
          If this fails, go to C
          query: Answer it in constant time
  end
C: (Graph Retirement Epoch)
  for $R = \lambda_2 n \log^2 n$ insertion-deletion operations do
  begin
    case of
          random insertion: Perform it by running e.g. the algorithm of ([10])
          random deletion: Perform it by running e.g. the algorithm of ([10])
          query: Answer it in constant time
  end
  flag := Total_Reconstruction
  if flag = "success" then go to B
    else go to C
  end
end (main loop)
end (Algorithm)
```

6 APPENDIX 2: Proofs of Theorems and Lemmas

6.1 Proof of Lemma 2:

Lemma 2: For each node w visited in Neighborhood Search:

$$\Pr\{\text{at least one pool edge out of } w \text{ exists}\} \geq \frac{1}{4}$$

Proof : By ([3]) the distribution of any degree of a vertex is *asymptotically Poisson*, right after a successful total reconstruction. Let $X_i(w)$ be the degree of *w in the pool.* Since $X_0(w)$ is Poisson (in the beginning of the activation period), then

$$E(X_0(w)) = Var(X_0(w))$$

It is known (see e.g. [3]) that:

$$\Pr\{X_0 = 0\} \leq \frac{\sigma^2}{\sigma^2 + \mu^2}$$

Thus:

$$\Pr\{X_0 = 0\} \leq \frac{1}{2}$$

i.e.

$$\Pr\{\text{at least one pool edge out of } w \text{ exists in the beginning of the current activation period}\} \geq \frac{1}{2}$$

Hence:

$$\begin{aligned} &\Pr\{\text{at least one pool edge out of } w \text{ exists at some step of the current activation period}\} \geq \\ &\geq \frac{1}{2}\Pr\{\text{edge is not in retirement}\} \qquad (4) \end{aligned}$$

In a subsequent section we prove that $\Pr\{\text{length of retirement period} \leq \lambda_1 \log n\} \geq 1 - n^{-\lambda_2}$. Also,

$$\begin{aligned} &\Pr\{\text{a particular edge which is not in the tree is not in retirement}\} = \\ &= 1 - \frac{\text{length of retirement period}}{\text{length of retirement period} + \text{length of period in the pool}} \end{aligned}$$

Edges are put in retirement only by random deletions. Since deletions are in random places and each retires only $k_1 \log n$ edges, we have:

$$\Pr\{\text{an edge in pool is going to retire in current deletion}\} \leq \frac{k_1 \log n}{(c - \alpha')n}$$

because the pool does not empty, due to the pipeline operation of reactivations. Let $f = \frac{k_1 \log n}{(c-\alpha')n}$. An edge *survives* a deletion if it is not retired. Because deletion operations are random and independent, $\Pr\{\text{a particular edge survives for } x \text{ consecutive operations}\} = (1-f)^x$. For $x = \lambda_1 \log n$ we get:

$$\begin{aligned} &\Pr\{\text{a particular edge survives for } x \text{ deletions in a row}\} = \\ &= (1-f)^x \geq 1 - xf \geq 1 - \frac{k_1 \log n \lambda_1 \log n}{(c-\alpha')n} \geq 1 - \frac{k_1 \lambda_1 \log^2 n}{n} \qquad (5) \end{aligned}$$

Let X_1 the event "length of retirement period $\leq \lambda_1 \log n$". Let Let X_2 the event "length of survival period $\geq \lambda_1 \log n$". Then, by (4), (5)

$$\begin{aligned} &\Pr\{\text{for a particular } w, \text{ at least one pool edge out of } w \text{ exists}\} \geq \\ &\geq \frac{1}{2}\Pr\{X_1\}\Pr\{X_2\} \geq \frac{1}{2}(1 - n^{-\lambda_2})\left(1 - \frac{k_1\lambda_1 \log^2 n}{n}\right) \geq \frac{1}{4} \end{aligned}$$

for all n sufficiently large. □

6.2 Proof of Lemma 3:

Lemma 3: $\Pr\{E_2|E_1\} \geq \frac{1}{4}$

Proof: An $e' = \{w, w'\}$ which reconnects the tree successfully must satisfy test (*) i.e. $d(w', R) < \alpha \log n$. The constant α is such that more than half of the nodes are in distance less than $\alpha \log n$ away from root R during an activation epoch of the graph. Let E be the set of all such nodes before reconnection. Since $d(u, R)$ is at least $(k_1 + \alpha) \log n$ (before removal of e), all nodes of E are in the piece in which w' belongs. Since w' was randomly selected (because e' was drawn from the pool), the probability that w' is in E is bigger than 1/2. But then $\Pr\{d(w', R) < \alpha \log n\} > \frac{1}{2}$ because $d(w', R)$ does not change due to reconstruction. □

6.3 Proof of Lemma 4:

Lemma 4: Each neighborhood search succeeds with probability at least $1 - n^{-\gamma}$. (γ a constant which can be made as large as we want by increasing k_1).

Proof: The probability that a coupled pair of phases of neighborhood searches out of u, v succeeds is at least (by Lemmas 2, 3)

$$\left(\frac{1}{4}\right)\left(\frac{1}{2}\right)\left(\frac{1}{2}\right) = \frac{1}{16}$$

The $k_1 \log n$ phases are during independent edges, say $e_i = \{w_i, w_i'\}$ (they are from the pool and the ends w_i' (of distinct edges e_i') are unrelated). Thus the probability of total failure of the neighborhood search is at most:

$$\left(1 - \frac{1}{16}\right)^{k_1 \log n} \leq n^{-\gamma}$$

where $\gamma \geq k_1 \log \frac{15}{16}$. □

6.4 Proof of Lemma 5:

Lemma 5: The number of successes in each neighborhood search is $\Theta(\log n)$ with high probability.

Proof: From Lemma 4 and the description of phase i we have that:
$\Pr\{$a coupled phase i (both sides) succeeds$\} \geq \frac{1}{16}$. Thus, for $k_1 \log n$ coupled phases, by Chernoff bounds we have that:

$$\forall \beta \in (0,1),\ \Pr\left\{\text{number of coupled successes} \geq (1-\beta)\frac{1}{16}k_1 \log n\right\} \geq 1 - n^{-k_2} \text{ for some } k_2$$

So with high probability we have $\Theta(\log n)$ successes in each neighborhood search, of which we select the one hooking one of the pieces closest to the root R. □

6.5 The case of deletions

Definition: The deletions which lead to Type-1 reconnection are called "incremental deletions". We have to consider only incremental deletions. Since the number of places of the giant component is ϵn, the number of u such that $d(u,R) \le (k_1+\alpha)\log n$ is $\le n$ anyway (type-1 reconnections) and deletions are random, we have

$$\Pr\{\text{an incremental deletion happens at a particular pair } u \text{ and } w',\ (w' \text{ a place})\} \le \frac{1}{n}\frac{1}{\epsilon n} \le \frac{1}{\epsilon n^2}$$

Type-1 reconnections become cumulative (with respect to increasing the diameter) only when the reconnecting pieces (each characterized by a pair $\{u_i, w_i'\}$) form a long "tail", i.e. equivalently only when the initial pair $\{u, w'\}$ of the first Type-1 reconnection is again selected to be continued by the second such pair. By Chernoff then, in the Bernoulli of $N_D \le (1+\beta)\frac{\lambda_1}{2}n^2\log n$ incremental deletions we have that

$$\Pr\{\text{each deletion increases the tail started at a place } (u,w')\} \le \frac{1}{\epsilon n^2}$$

so we get, for $\beta' \in (0,1)$, $\sigma' > 1$,

$$\Pr\left\{\text{number of incremental deletions accumulated} \le \sigma'(1+\beta')\left(\frac{(1+\beta)\lambda_1}{2}n^2\log n\right)\left(\frac{1}{\epsilon n^2}\right)\right\} \le$$
$$\le \exp\left(-\frac{\beta'^2}{2}\sigma'\frac{(1+\beta)\lambda_1}{2\epsilon}\log n\right)$$

Let $\gamma_2 = \frac{\beta'^2}{2}\sigma'\frac{(1+\beta)\lambda_1}{2\epsilon}$. Thus, since $\Delta_D \le \log n \times$ (number of incremental deletions which are cumulative at the same pair), we get

$$\Pr\left\{\Delta_D \ge \sigma'(1+\beta')(1+\beta)\frac{\lambda_1}{2\epsilon}\log^2 n\right\} \le n^{-\gamma_2}$$

6.6 Proof of Corollary 1:

Corollary 1: During the whole activation period, the time for an operation is $\Theta(\log^3 n)$ with probability $\ge 1 - n^{-\gamma_4}$.
Proof: The most costly operation is a deletion and subsequent successful reconnection of the giant tree, which takes $O(\log^3 n)$ time and by Theorem 1 all such operations are executed while the induction hypothesis is holding (i.e. with probability $\ge 1-n^{-\gamma_4}$ by Theorem 1). □

6.7 Proof of Theorem 2

Theorem 2: Conditioning on the fact that the graph undergoes a graph activation epoch and provided that the pool is reactivated successfully, each operation in the epoch takes at most $O(\log^3 n)$ time, with probability $\ge 1 - n^{-\delta}$.
Proof: Let $\delta = \gamma_4$. Note also that:

- case b1 takes $O(1)$ time.
- case b2 takes $O(\log n)$ time (we are in a graph activation epoch).
- case c1 takes $O(1)$ time.
- case c2 takes $O(\log n)$ time (we are in a graph activation epoch).

□

6.8 Proof of Lemma 7:

Lemma 7: For each edge x which was used and failed (except for midget edges) there exist $\lambda_1 > 2$, $\lambda_2 > 2$ such that with probability $\geq 1 - n^{-\lambda_2}$

$$retirement - interval(x) \leq \lambda_1 \log n$$

where $retirement - interval(x)$ is the length of the retirement interval for edge x.

Proof: We do not consider here the cases where $B < \log n$. Used edges which had failed on such small components are not reactivated because most probably the small components do not reconnect to T and such edges are of no subsequent use in the pool during a graph activation epoch.

Let thus $|B| > \log n$. By Lemma 4, the neighborhood search out of the pair u, v (of the deleted edge e) succeeds with high probability (in fact it succeeds in $O(\log^3 n)$ time with probability $\geq 1 - n^{-\gamma}$). Note that during the neighborhood search the pool edges used are the edges from the pool at time just after A, B were separated by e.

Conditioning on the success of neighborhood-search(u, v) (i.e. after A, B reconnect by a pool edge) let $y + 1$ be the number of edge deletions such that the last of them first breaks the giant tree T.

Because T is giant during the graph activation epoch, a random deletion occurs in the tree T (and so breaks it) with probability

$$\frac{|T|}{\text{number of edges in trees}} > g$$

where $g > 0$ a constant. (Note, of course, that the entire graph may not be disconnected in this case).
Thus, $\forall \lambda_3' \geq 1$

$$\Pr\{y \geq \lambda_3' \log n\} \leq \left(1 - \frac{|T|}{\text{number of edges in trees}}\right)^{\lambda_3' \log n} < (1-g)^{\lambda_3' \log n} = n^{-\lambda_3}$$

with $\lambda_3 = \lambda_3' \log\left(\frac{1}{1-g}\right)$. Take a sequence L of $|L| = \lambda_1 \log n$ operations and choose $\forall \beta \in (0,1)$, $\lambda_3 > \frac{\lambda_1}{2}(1+\beta)$. By Chernoff bounds

$$\Pr\{L \text{ contains at least } \lambda_3 \log n \text{ deletions}\} \geq 1 - \exp\left(-\frac{\beta^2}{2}\lambda_1 \log n\right) \geq 1 - n^{-\frac{\beta^2}{2}\lambda_1}$$

Thus, if $\lambda_2 = \min\left(\lambda_3, \frac{\beta^2}{2}\lambda_1\right)$, we have

$$\Pr\{\text{retirement-interval}(x) \leq \lambda_1 \log n\} \geq 1 - n^{-\lambda_2}$$

In Case 2, x is a negative$-A$ edge. Then edge x is reactivated after A, B reconnect (by a pool edge or insertion) and after the first deletion again disconnects T. The analysis is symmetric to Case 1. In Case 3 x is a positive edge. Let x' be the edge which reconnected A, B in the tree in the neighborhood search where x failed. Let B_1 be the part of the tree defined by the fundamental cycles of x and x' and A_1 the rest. The situation now is analogous to Cases 1, 2 (i.e. a deletion is needed to break B_1 or not, equiprobably). □

6.9 Proof of Theorem 3

Theorem 3: Our algorithm has $O(\log^3 n)$ amortized cost per operation, with high probability (or amortized expected) in the sense that a long sequence of M operations takes expected time $O(M \log^3 n)$, for large enough M.

Proof: Condition on the events of the length of a graph activation period being $O(\lambda_1 n^2 \log n)$ and of Theorem 2, also on the event of Lemma 5. Let $\lambda_5 = \max\{\lambda_4, \delta, k_1\}$. Then with probability $\geq 1 - n^{-\lambda_5}$ the amortized operation time is at most

$$\frac{A\,\Theta(\log^3 n) + R\,o(n)}{A+R} = \frac{\lambda_1 n^2 \log n\,\Theta(\log^3 n) + \lambda_2 n \log^2 n\,o(n)}{A+R} = \Theta(\log^3 n)$$

So, the expected operation time over any sequence of M operations which includes multiples of A, R operations, is:

$$\text{Expected Operation Time } = (1 - n^{-\lambda_5})\Theta(\log^3 n) + n^{-\lambda_5} O(n) \leq \Theta(\log^3 n)$$

□