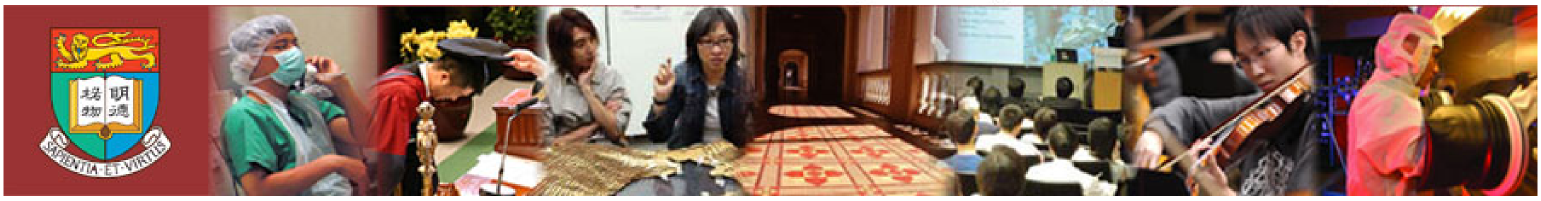

Dynamic Tree Shortcut with Constant Degree

T-H. Hubert Chan[1], Xiaowei Wu[1], Chenzi Zhang[1], and Zhichao Zhao[1]

The University of Hong Kong
{hubert,xwwu,czzhang,zczhao}@cs.hku.hk

Abstract. Given a rooted tree with n nodes, the tree shortcut problem is to add a set of shortcut edges to the tree such that the shortest path from each node to any of its ancestors is of length $O(\log n)$ and the degree increment of each node is constant. We consider in this paper the dynamic version of the problem, which supports node insertion and deletion. For insertion, a node can be inserted as a leaf node or an internal node by sub-dividing an existing edge. For deletion, a leaf node can be deleted, or an internal node can be merged with its single child. We propose an algorithm that maintains a set of shortcut edges in $O(\log n)$ time for an insertion or deletion.

1 Introduction

The problem of adding a set of shortcut edges S to a tree $T(V,E)$ to reduce the *hop-diameter* of the resulting graph $G(V, E \cup S)$ has been studied for many years [17, 1, 5, 4, 15, 12]. We call a simple path P from node u to v in G *straight* if the sequence of nodes in P is a sub-sequence of the unique path from u to v in T. The hop-diameter of the graph is the maximum number of edges in the shortest straight path between any two nodes in the graph. Given a rooted tree with n nodes, the hop-diameter of the tree can be as large as $\Theta(n)$. In applications like index-based search or broadcast in tree-networks, the hop-diameter (or the height) of the tree is crucial to the performance. To improve the efficiency, researchers have been analysing how to add shortcut edges between ancestor and descendant such that the hop-diameter of the resulting graph is small (i.e., $\log n$). Note that the hop-diameter can be easily reduced to $O(1)$ by adding a shortcut edge between each pair of ancestor and descendant in the tree. However, the resulting graph has maximum degree $\Omega(n)$. Hence another objective on shortcutting a tree is the *degree increment* on each node $u \in V$, which is the number of edges in S incident to u. We refer to the problem of shortcutting a tree as the tree-shortcut problem, which has application to spanners [7, 11].

A natural extension of the tree-shortcut problem is to support operations on the tree [10]. In the dynamic setting, the tree structure will be changed in each iteration (by one operation) and we need to maintain the set of shortcut edges such that the hop-diameter and the degree increment on each node are still bounded. We refer to the dynamic version of the tree-shortcut problem as the dynamic-tree-shortcut problem. We consider in this paper the dynamic-tree-shortcut problem that supports two kinds of operations: node insertion and deletion. The insertion operation inserts a node to the tree as a leaf node or as an internal node by sub-dividing an existing edge. The deletion operation deletes a leaf node (together with the incident edge) or an internal node that has a single child (the internal node is merged with its child after the deletion).

1.1 Related works

Yao [17] studied a special version of tree-shortcut problem that has application to range query, in which the given tree is a single chain with n nodes. It is shown in [17] that we can add $\Theta(n)$ shortcut edges to guarantee $O(\alpha(n))$ hop-diameter, where $\alpha(n)$ is the inverse Ackermann function introduced by Tarjan [13]. Note that the maximum degree increment on each node in their construction is $\Omega(n^{\frac{1}{\alpha(n)}})$. Their hop-diameter is proved to be asymptotically optimal, if the total number of shortcut edges is $\Theta(n)$. Their result was later generalized to arbitrary trees by Chazelle [5] and Thorup [15] such that $\Theta(n)$ shortcut edges are added to a tree with n nodes to obtain $O(\alpha(n))$ hop-diameter and $\Omega(n^{\frac{1}{\alpha(n)}})$ degree increment. Conjecture on a generalized version of the tree-shortcut problem on directed graphs were also made [14], but it was later disproved [8, 9]. Solomon and Elkin [12] considered the trade-off between degree increment and hop-diameter of the tree-shortcut problem, and proposed an algorithm that guarantees $O(k)$ degree increment on each node and $O(\log_k n + \alpha(k))$ hop-diameter by adding $O(n)$ shortcut edges to the tree, where k is any integer less than n. Note that [12] obtained, for the first time, constant degree increment and $O(\log n)$ hop-diameter for the tree-shortcut problem. However, operations such as node insertion and deletion are not supported by their data structure.

As a parallel line of research, the dynamic-tree-shortcut problem is also extensively studied. Sleator and Tarjan [10] derived a dynamic data structure that can be used to shortcut a tree such that the degree increment on each node and the hop-diameter are both bounded by $O(\log n)$. Moreover, they showed that the data structure can be maintained in $O(\log n)$ time against operations such as cutting a tree into two by removing one edge, or linking two trees by adding an edge. Note that node insertion and deletion are special cases of the above operations and hence are also supported by [10]. A centroid decomposition of the tree and a biased search tree [3] on each centroid path were maintained in [10]. We adopt similar ideas on constructing shortcut edges for the dynamic-tree-shortcut problem and improve their result by reducing the degree increment to a constant.

1.2 Our Contribution

Given a rooted tree with n nodes, we construct in this paper a dynamic data structure for shortcut edges that guarantees $O(1)$ degree increment on each node and $O(\log n)$ hop-diameter. Moreover, we show that our data structure can be maintained against node insertions and deletions in $O(\log n)$ time by adding or deleting $O(\log n)$ shortcut edges after each operation. We summarize and compare some related results as follows.

	Degree-increment	Hop-diameter	Update time for node insertion/deletion
[5, 15]	$\Omega(n^{1/\alpha(n)})$	$O(\alpha(n))$	Not supported
[12]	$O(k)$	$O(\log_k n + \alpha(k))$	Not supported
[10]	$O(\log n)$	$O(\log n)$	$O(\log n)$
This paper	$O(1)$	$O(\log n)$	$O(\log n)$

In the table, n is the number of nodes in the tree (before node insertion/deletion), k is any integer less than n and $\alpha(n)$ is the inverse Ackermann function of n.

2 Preliminaries

Given a rooted tree $T(V, E)$, the *distance* $d(u, v)$ between two nodes $u, v \in V$ is the number of edges in the unique path from u to v in the tree. For each internal node $u \in V$, let Child(u) be the set of children of u in the tree. We use $[n]$ to denote $\{1, 2, \ldots, n\}$ for any positive integer n and $\log n$ to denote $\log_2 n$.

Definition 1 (Operation). *There are two kinds of operations on the tree.*

- **Insertion.** *Operation $(x, insertion)$ inserts a new node x to the tree $T(V, E)$. The new node x can be inserted as a child of an existing node or an internal node that subdivides an existing tree edge. Note that x indicates a position in the new tree.*
- **Deletion.** *Operation $(x, deletion)$ deletes an existing node x, which is either a leaf node or an internal node with only one child, from the tree $T(V, E)$. If x is a leaf node, then the edge incident to x is also deleted; otherwise x is merged with its single child. Note that x specifies a node in the original tree.*

Definition 2 (Shortcut Edge). *In a rooted tree, a* shortcut edge *is an edge between a node and one of its descendants.*

Let S be a set of shortcut edges built on $T(V, E)$. Let $G(V, E \cup S)$ be the graph obtained by adding the shortcut edges in S to the tree $T(V, E)$. In the dynamic setting, we always use G and T to denote the current graph and tree (after the operations and updates).

For each edge $e = (u, v) \in E \cup S$, let $l_e = d(u, v)$ be its *length.* Note that $l_e = 1$ for all $e \in E$. A path (from node u_1 to node u_q) is denoted by a sequence of nodes $P = (u_1, u_2, \ldots, u_q)$. The *length* of path $P = (u_1, u_2, \ldots, u_q)$ is the sum of lengths of the edges in the path, which is $\sum_{i=1}^{q-1} l_{(u_i, u_{i+1})} = \sum_{i=1}^{q-1} d(u_i, u_{i+1})$. With a slight abuse of notation, we use $|P|$ to denote the number of nodes in P and $u \in P$ to denote that $u \in V$ is in the path.

Definition 3 (Straight Path). *A path $P = (u_1, u_2, \ldots, u_q)$ is* straight *iff its length equals the distance $d(u_1, u_q)$ between u_1 and u_q.*

Definition 4 (Hop-distance, Hop-diameter, Degree). *The* hop-distance $h_G(u,v)$ *between two nodes* $u, v \in V$ *in* G *is the minimum number of edges in a straight path from* u *to* v. *The* hop-diameter $\varDelta_G = \max_{u,v\in V}\{h_G(u,v)\}$ *of* G *is the maximum hop-distance between any two nodes. Let* $\mathsf{deg}_S(u)$ *be the number of shortcut edges incident to* $u \in V$. *We call* $\mathsf{deg}_S(u)$ *the* degree *of* u. *The maximum degree in* G *is denoted by* $\mathsf{deg}_S := \max_{u\in V}\{\mathsf{deg}_S(u)\}$.

Theorem 1. *A set of shortcut edges* S *for a rooted tree* $T(V,E)$ *of* n *nodes can be found such that graph* $G(V, E\cup S)$ *ensures* $\varDelta_G = O(\log n)$ *and* $\mathsf{deg}_S = O(1)$. *For each operation on* T, *shortcut edges* S *can be maintained in* $O(\log n)$ *time, where* n *is the number of nodes before the operation.*

We prove Theorem 1 in the rest of this paper by providing a data structure in Section 3, and an algorithm in Section 4 to maintain the data structure against operations. The following definitions are important for the construction and the update algorithm. Let T_u be the sub-tree rooted at node u. We denote by $|T_u|$ the number of nodes in T_u.

Definition 5 (Heavy Child Mapping). *For any internal node* u, *a child* x *of* u *is* heavy *iff* $4|T_x| \geq \max_{y\in Child(u)}\{|T_y|\}$. *A mapping* f *that maps each internal node* u *to one of its children is called a* heavy child mapping *iff* $f(u)$ *is a heavy child of* u *for all internal node* $u \in V$.

Note that the heavy child mapping may be non-unique. For each internal node u we set $f(u) = \arg\max_{x\in \mathrm{Child}(u)}\{|T_x|\}$ when the tree $T(V,E)$ is given at the beginning. We may change the mapping f in the update algorithm. However, we always maintain f as a heavy child mapping after each update. Based on the heavy child mapping $f(u)$, let $\hat{w}_u = |T_u| - |T_{f(u)}|$ for each internal node u and $\hat{w}_u = 1$ for each leaf node u.

Definition 6 (Proper Weighting). *A weighting function* w *that assigns an integer weight* w_u *to each node* $u \in V$ *is a* proper weighting *iff* $w_u \in [\frac{\hat{w}_u}{4}, 4\hat{w}_u]$.

Note that a proper weighting is based on a heavy child mapping. The proper weighting on all nodes is also not unique. We set $w_u = \hat{w}_u$ for each node $u \in V$ when the tree $T(V,E)$ is given at the beginning. We may change the weights of nodes in the update algorithm. However, we always maintain the proper weighting w after each update. Given a path $P = (u_1, u_2, \ldots, u_q)$ and a proper weighting w, let $w(P) = \sum_{i=1}^{q} w_{u_i}$. Unless otherwise specified, we assume that a heavy child mapping f and a proper weighting w are maintained throughout this paper.

3 The Structure of Shortcut Edges

To build shortcut edges on the tree, we first break the tree into *centroid paths* and then build shortcut edges within each centroid path. We further break a centroid path into *buckets*, each of which contains $O(\log n)$ consecutive nodes. We build Static-Path-Shortcut (Section 3.1) on each bucket and Dynamic-Path-Shortcut (Section 3.2) between buckets using biased-skip-list. Due to page limit, missing proofs are provided in Appendix A.

Centroid Paths. For each internal node $u \in V$, we call $(u, f(u))$ a *joint*. Let $E_f = \{(u, f(u)) | u \text{ is an internal node in } V\}$ be the set of all joints. We call the edges in $E \backslash E_f$ *links*. Given a rooted tree $T(V, E)$ and a heavy child mapping f, we partition the tree into paths $P_1, P_2, \dots, P_r$ by removing all *links*. We call each of those paths a *centroid path*, and the set of centroid paths the *Tree-Decomposition* of T. The set of centroid paths can be induced by $T_f(V, E_f)$. For all $i \in [r]$, we have $P_i = (x, f(x), f(f(x)), \dots)$ for some $x \in V$. We call x the *top-node* of P_i.

Claim 1. *The nodes in the path P from any node $u \in V$ to the root of the tree are contained in at most $O(\log n)$ centroid paths.*

Proof. Note that P consists of joints and links. For each link (x, y) in P such that $x \in \text{Child}(y)$, we have $x \neq f(y)$. By definition of heavy child mapping f, we have $|T_y| \geq \frac{5}{4}|T_x|$, which means that the number of links in P is $O(\log n)$. Since all consecutive joints in P are contained in the same centroid path, the nodes in P are contained in at most $O(\log n)$ centroid paths. $\square$

3.1 Static Path Shortcut

In this section, we consider how to add shortcut edges to a path. Given P and w, the Static-Path-Shortcut defined as follows is a set of shortcut edges and must be unique. We can run Alg. 1 as $\text{SPS}(P, 1, q, t)$ to construct it.

Definition 7 (Static-Path-Shortcut). *Given a path $P = (u_1, u_2, \dots, u_q)$ and a proper weighting w, let $t_k = \sum_{i \in [k]} w_{u_i}$ for all $k \in [q]$ and $t_0 = 0$. If $q \leq 2$, then the* Static-Path-Shortcut *of P is an empty set; otherwise the Static-Path-Shortcut of P contains the edge (u_1, u_m), where $t_{m-1} < \frac{t_q}{2} \leq t_m$, and the edges in the Static-Path-Shortcuts of sub-paths $(u_2, \dots, u_{m-1})$ and $(u_{m+1}, \dots, u_q)$. A sub-path $(u_i, \dots, u_j)$ is empty if $j < i$.*

Lemma 1. *Given a path $P = (u_1, u_2, \dots, u_q)$ and proper weighting w, its Static-Path-Shortcut S can be constructed in $O(q)$ time such that $\mathsf{deg}_S = O(1)$ and $h_G(u_1, u_i) = O(\log \frac{w(P)}{w_{u_i}})$ for $i \in [q]$, where we consider P as a tree rooted at u_1.*

Algorithm 1 $\text{SPS}(P, i, j, t)$:

```
Input: Path $P = (u_1, u_2, \dots, u_q)$, positions $1 \leq i < j \leq q$ and cumulative weights $t$.
1: if $i + 1 < j$ then
2:     $\tau \leftarrow \frac{t_j + t_{i-1}}{2}$;
3:     find integer $l \geq 0$ such that $t_{i+2^l-1} \leq \tau \leq t_{i+2^{l+1}-1}$ or $t_{j-2^{l+1}+1} \leq \tau \leq t_{j-2^l+1}$.
4:     find $m$ such that $t_{m-1} < \tau \leq t_m$ in the above range using binary search.
5:     return $\{(u_i, u_m)\} \cup \text{SPS}(P, i+1, m-1, t) \cup \text{SPS}(P, m+1, j, t)$
```

3.2 Dynamic Path Shortcut

Definition 8 (Path-Decomposition). *Given a path $P = (u_1, u_2, \dots, u_q)$, a* Path-Decomposition *of P is a sequence of* buckets $B_1, B_2, \dots, B_k$ *such that*

$B_i = (u_{b_{i-1}+1}, u_{b_{i-1}+2}, \ldots, u_{b_i})$ for all $i \in [k]$, where $0 = b_0 < b_1 < b_2 < \ldots < b_k = q$. The number of nodes in each bucket B_i, denoted by $|B_i| = b_i - b_{i-1}$, satisfies $2\log(4n) \le |B_i| \le 10\log(4n)$. For the case when $|P| < 2\log(4n)$, we set $k = 1$ and the bucket size requirement is removed.

Note that the Path-Decomposition of a path P may be non-unique. We ensure $4\log(4n) \le |B_i| \le 8\log(4n)$ when the tree is given for the first time but only maintain $2\log(4n) \le |B_i| \le 10\log(4n)$ in the update algorithms. If $|P| \le 8\log(4n)$, we use one bucket to contain all nodes. We consider a bucket as a sub-path of the path P. We use $u \in B_i$ to denote that u is in bucket B_i. Given a proper weighting w, let $w(B_k) = 1$ and $w(B_i) = \sum_{u \in B_{i+1}} w_u$ be the *weight* of B_i, for $i \in [k-1]$. Note that by definition, the weight of a bucket is the total weight of nodes in the *next* bucket. The reason behind this definition will be clear in the proof of Lemma 2.

Definition 9 (Biased-Skip-List [2]). *Given a sequence of buckets $B_1, B_2 \ldots, B_k$ and a proper weighting w, a* biased-skip-list *h assigns an integer* height *h_i to each bucket B_i such that (1) $\lfloor \log w(B_i) \rfloor \le h_i \le \log(4n)$ for all $i \in [k]$; (2) $h_1 = h_k = h_{\max} = \max_{i \in [k]}\{h_i\}$.*

Definition 10 (Successor Pointer). *Given a biased-skip-list h, for each height $h \le h_{\max}$, buckets of height at least h are kept in sorted order in a doubly linked list called h-list. The* successor pointer *$s_{i,h}$ points to the successor of B_i in h-list.*

Recall the following algorithms [2] of direct search and finger search.

Algorithm 2 Direct_Search(h, s, x):

```
Input: biased-skip-list $h$, successor pointers $s$ and target bucket $B_x$.
1: $i \leftarrow 1$; $j \leftarrow h_{\max}$;
2: while $i \ne x$ do
3:     if $s_{i,j} \le x$ then
4:         $i \leftarrow s_{i,j}$;
5:     else if $i < x$ then
6:         $j \leftarrow j - 1$;
```

Algorithm 3 Finger_Search(h, s, x, y):

```
Input: biased-skip-list $h$, successor pointers $s$, start bucket $B_x$ and target bucket $B_y$.
1: $i \leftarrow x$; $j \leftarrow h_x$;
2: while $i \ne y$ do
3:     if $j < h_i$ AND $s_{i,j+1} \le y$ then                 ▷ Up phase.
4:         $i \leftarrow s_{i,j+1}$; $j \leftarrow j + 1$;
5:     else if $s_{i,j} \le y$ then
6:         $i \leftarrow s_{i,j}$;
7:     else if $i < y$ then                                    ▷ Down phase.
8:         $j \leftarrow j - 1$;
```

Fact 1 ([2]). *Given a biased-skip-list, the time complexity for direct search of bucket B_x is $O(\log \frac{w(P)}{w(B_x)})$ and the time complexity for finger search is $O(\log w(P))$. The time complexity to maintain the biased-skip-list(s) is $O(\log W)$ for the following operations, where $W = \max\{W_a, W_b\}$, W_a and W_b are the total weights of buckets before and after the operation, respectively: (1) adding, deleting or re-weighting one bucket. (2) splitting a biased-skip-list into two. (3) merging two biased-skip-lists into one.*

Definition 11 (Dynamic-Path-Shortcut). *Given a Path-Decomposition $B_1, B_2, \ldots, B_k$ of path $P = (u_1, u_2, \ldots, u_q)$ and a biased-skip-list h on the buckets, the* Dynamic-Path-Shortcut *is a set of shortcut edges that contains*

- *edges in the Static-Path-Shortcut of each bucket B_i,*
- *$(u_{b_{i-1}+1}, u_{b_i-h_i+1})$ and $(u_{b_i-h_i+1}, u_{b_i})$ for all $i \in [k]$,*
- *$(u_{b_i-t+1}, u_{b_j-t+1})$ and $(u_{b_i-t+2}, u_{b_j-t+1})$ for all i, j, t such that $2 \le t \le h_i$ and $B_j = s_{i,t}$,*
- *$(u_{b_i}, u_{b_{i+1}})$ for all $i \in [k-1]$.*

Lemma 2. *Given a path P with proper weighting w, its Dynamic-Path-Shortcut S guarantees $\mathsf{deg}_S = O(1)$, $h_G(u_1, u_i) = O(\log \frac{w(P)}{w_{u_i}})$ and $h_G(u_i, u_j) = O(\log(4n))$ for all $u_i, u_j \in P$.*

Lemma 3. *To maintain the Dynamic-Path-Shortcut(s) of path P, the time complexity for the update algorithms is $O(\log(4n))$ for the following operations.*

- *Adding, removing or re-weighting (while maintaining a proper weighting) a node $u \in P$.*
- *Splitting the path P into two paths by deleting one edge in P.*
- *Concatenate a path P', with its Dynamic-Path-Shortcut, to the end of P.*

3.3 Dynamic Tree Shortcut

Definition 12 (Dynamic-Tree-Shortcut). *Given a rooted tree $T(V, E)$, a heavy child mapping f and a proper weighting w, the* Dynamic-Path-Shortcut *of T contains all edges in the Dynamic-Path-Shortcut of each centroid path P_i in the Tree-Decomposition $P_1, P_2, \ldots, P_r$ of T.*

Lemma 4. *Given a rooted tree T of n nodes, the Dynamic-Path-Shortcut S guarantees $\mathsf{deg}_S = O(1)$ and $h_G(u, v) = O(\log n)$ for each ancestor-descendant pair (u, v).*

Proof. First observe that $\mathsf{deg}_S = O(1)$ by Lemma 2 and the disjunction of centroid paths.

Let P be the tree path from node v to its ancestor u, which is a sub-path of the tree path from v to the root. By Claim 1, nodes in P are contained in $k = O(\log n)$ centroid paths. Let $P'_1, P'_2, \ldots, P'_k$ be those centroid paths such that $u \in P'_1$, $v \in P'_k$ and the parent of the top-node of P'_{i+1} is contained in P'_i, for all $i \in [k-1]$.

Let x_i be the top-node of P'_i, for all $i \in [k]$. For all $i \in [k-1]$, let $y_i \in P'_i$ be the parent of x_{i+1}. Let $y_k = v$. Hence P follows $v = y_k \rightsquigarrow (x_k, y_{k-1}) \rightsquigarrow (x_{k-1}, y_{k-2}) \rightsquigarrow \ldots \rightsquigarrow (x_2, y_1) \rightsquigarrow u$. By definition of proper weighting, for all $1 < i \leq k$ we have

$$w(P'_i) \leq 4|T_{x_i}| \leq 4(|T_{y_{i-1}}| - |T_{f(y_{i-1})}|) = 4\hat{w}_{y_{i-1}} \leq 16 w_{y_{i-1}}.$$

Instead of following tree path P, by Lemma 3, there exists a constant c such that we can use the Dynamic-Path-Shortcut of each centroid path to reach x_i from y_i by a straight path of length $c \log \frac{w(P'_i)}{w_{y_i}}$ for all $1 < i \leq k$, and reach u from y_1 by a straight path of length $O(\log n)$.

Hence the hop-distance between v and u using Dynamic-Path-Shortcut is

$$\begin{aligned} h_G(u,v) &\leq O(\log n) + \sum_{i=2}^{k} (c \log \frac{w(P'_i)}{w_{y_i}} + 1) \leq O(\log n) + k + c \sum_{i=2}^{k} (\log \frac{16 w_{y_{i-1}}}{w_{y_i}}) \\ &\leq O(\log n) + (4c+1)k + c \log \frac{w_{y_1}}{w_v} = O(\log n). \end{aligned}$$

□

4 Update Algorithm for Operations

We have introduced how to build shortcut edges on a given tree such that the degree increment and the diameter are bounded after the construction. In this section, we will describe how to update the Dynamic-Tree-Shortcut after each operation $(x, \text{insertion/deletion})$ by giving update algorithms. The real challenge for maintaining the Dynamic-Tree-Shortcut is that the heavy child mapping f and proper weighting w, if not updated accordingly, may not hold due to the change of the number of nodes. Throughout this section, we assume the current number of nodes n in the tree is large enough.

By Definition 12, as long as the heavy child mapping f, the proper weighting w and the Dynamic-Path-Shortcut of each centroid path are maintained, the Dynamic-Tree-Shortcut is maintained as Lemma 4. For the maintainance of Dynamic-Path-Shortcut, the most crucial part is to maintain the buckets. We define three statuses of an object: *intact*, *safe* and *risky*, where an object is a bucket or a node or a joint. We call an object *maintained* iff it is in one of those three statuses. Note that the status of an object may change after each operation.

	intact	safe	risky
Bucket B	$\frac{\lvert B\rvert}{\log(4n)} \in [4, 8]$	$\frac{\lvert B\rvert}{\log(4n)} \in [3, 4) \cup (8, 9]$	$\frac{\lvert B\rvert}{\log(4n)} \in [2, 3) \cup (9, 10]$
Node u	$\frac{w_u}{\hat{w}_u} = 1$	$\frac{w_u}{\hat{w}_u} \in [\frac{1}{2}, 1) \cup (1, 2]$	$\frac{w_u}{\hat{w}_u} \in [\frac{1}{4}, \frac{1}{2}) \cup (2, 4]$
Joint $(u, f(u))$	$\frac{\lvert T_{f(u)}\rvert}{\max_{v \in \text{Child}(u)} \lvert T_v\rvert} = 1$	$\frac{\lvert T_{f(u)}\rvert}{\max_{v \in \text{Child}(u)} \lvert T_v\rvert} \in [\frac{1}{2}, 1)$	$\frac{\lvert T_{f(u)}\rvert}{\max_{v \in \text{Child}(u)} \lvert T_v\rvert} \in [\frac{1}{4}, \frac{1}{2})$

For the case when a centroid path P is of size $|P| \leq 10 \log(4n)$, since one bucket B is sufficient to contain the whole path, we call the bucket B intact if $|B| \leq 8 \log(4n)$; safe if $8 \log(4n) < |B| \leq 9 \log(4n)$ and risky if $9 \log(4n) < |B| \leq$

$10\log(4n)$. By the above definitions, at the beginning when the tree is given, the Dynamic-Tree-Shortcut we construct in Section 3 ensures that all buckets, joints and nodes are **intact**.

Due to page limit, the update algorithms for the reconstruction of risky buckets, risky joints and risky nodes are deferred to Appendix C. All those algorithms rebuild a risky object and turn it intact.

We give the following algorithm to update the Dynamic-Tree-Shortcut for each operation $(x, \text{insertion/deletion})$. Given an operation $(x, \text{insertion/deletion})$, we call a joint $(u, f(u))$ *touched* if u is an ancestor of x; a node u *touched* if u is in the path P from x to the root and $f(u) \notin P$. Let $k_1 = 18$ and $k_2 = 36$ be constant parameters.

Algorithm 4 Update(x, insertion/deletion)

```
1: insert or delete a node at position $x$
2: update the Dynamic-Path-Shortcut of the centroid path containing $x$ ▷ Lemma 3
3: $n \leftarrow n + 1$ if $x$ is inserted; $n \leftarrow n - 1$ if $x$ is deleted
4: rebuild an arbitrary risky bucket (if exist)
5: rebuild $k_1$ touched risky joints that are closest to $x$ (if exist)
6: rebuild $k_2$ touched risky nodes that are closest to $x$ (if exist)
```

Lemma 5. *Given a Dynamic-Tree-Shortcut of a rooted tree such that all buckets, joints and nodes are intact, Alg. 4 keeps all buckets, joints and nodes maintained for any sequence of operations.*

We prove Lemma 5 by analysing the buckets, joints and nodes one by one as follows. The analysis of maintaining nodes is in Appendix B.

4.1 Maintaining the Buckets

Lemma 6. *Alg. 4 keeps all buckets maintained for any sequence of operations, if initially all buckets are intact.*

Proof. First we show that it takes more than $\frac{n}{2}$ operations to turn a bucket from intact to risky, or from safe to not-maintained, where n is number of nodes in the tree before those operations. Notice that if the size of a bucket is changed after operation $(x, \text{insertion/deletion})$, then it must be in the centroid path containing x. By Alg. 4 line 2 and Lemma 3, the bucket will be reconstructed and become intact. Hence the only case a bucket B changes from intact to risky is due to the change of the number of nodes in the tree, while the size of the bucket remains unchanged. Let n' be the number of nodes in the tree when bucket B becomes risky. Then we have either $4\log(4n) \le |B| < 3\log(4n')$ or $9\log(4n') < |B| \le 8\log(4n)$. In the first case we have $n' - n > n^{4/3} - n = n(n^{1/3} - 1) > n > \frac{n}{2}$ and in the second case we have $n - n' > n - n^{8/9} = n(1 - n^{-1/9}) > \frac{n}{2}$. Since n is changed by 1 after each operation, the number of operations needed to change a bucket from intact to risky is more than $\frac{n}{2}$. Similar argument can be applied to show that the number of operations needed to change a bucket from safe to not-maintained is also more than $\frac{n}{2}$.

Assume the contrary of Lemma 6 and let bucket B be the first bucket that is not maintained. Consider the last moment t when B is safe and let n^* be the number of nodes in the tree at moment t. Since each safe or risky bucket at moment t must be of size at least $2\log(4n^*)$, the number of safe or risky buckets is at most $\frac{n^*}{2\log(4n^*)}$. Consider moment $t+\frac{n^*}{2}$, after $\frac{n^*}{2}$ operations. Note that at this moment, all intact buckets at moment t and buckets created after moment t must not be risky. Moreover, bucket B must be risky at this moment, by the above analysis. However, since B is risky between moment t and $t+\frac{n^*}{2}$, a risky bucket other than B must be rebuilt after each of the past $\frac{n^*}{2}$ operations, which is a contradiction since there are at most $\frac{n^*}{2\log(4n^*)}$ risky buckets between moment t and moment $t+\frac{n^*}{2}$. □

4.2 Maintaining the Joints

Note that the status of a joint $(u, f(u))$ will not change if the sizes of all subtrees of u remain unchanged. Hence we only need to maintain the joints that are touched, after each operation. For each joint $(u, f(u))$, between its two consecutive rebuilds, we call the last moment before $(u, f(u))$ is changed from intact to safe the *marginal moment* of $(u, f(u))$.

Definition 13 (Class). *We put joints into* classes C_i, $i \in [\log n]$, *right after their marginal moments. A joint* $(u, f(u))$ *is put into* C_i *iff* $|T_{f(u)}| \in [2^i, 2^{i+1})$. *A joint* $(u, f(u)) \in C_i$ *is removed from* C_i *if it is touched by* 2^{i-1} *operations after its marginal moment.*

We show that joints in classes are either safe or intact. Since risky joints may be rebuilt, each joint can be put into and removed from classes multiple times.

Lemma 7. *For all* $i \in [\log n]$, *each joint* $(u, f(u)) \in C_i$ *is either safe or intact.*

Proof. It suffices to show that $(u, f(u)) \in C_i$ will become risky only after being touched by more than 2^{i-1} operations. Consider the marginal moment of $(u, f(u))$. Since $(u, f(u))$ is intact at this moment, we have $|T_{f(u)}| \geq |T_x|$ for all $x \in \mathrm{Child}(u)$. Let $t = |T_{f(u)}| \in [2^i, 2^{i+1})$ at this marginal moment.

Now consider the time when $(u, f(u))$ becomes risky. Let $t' = |T_{f(u)}|$ at this moment. Since $(u, f(u))$ is risky, there exists $y \in \mathrm{Child}(u)$ such that $y \neq f(u)$ and $|T_y| > 2t'$. Note that at the marginal moment of $(u, f(u))$, we have $|T_y| \leq t$, as argued above. Since only one node is inserted or deleted in each operation, the number of operations between the above two moments, is more than $|t - t'| + |t - 2t'| = t(|1 - \frac{t'}{t}| + |1 - 2\frac{t'}{t}|) \geq \frac{t}{2} \geq 2^{i-1}$. □

Lemma 8. *Each operation touches at most* 4 *joints in each class.*

Proof. Fix one operation and one class C_i. Consider any joint $(u, f(u)) \in C_i$. Similar to the proof of Lemma 7, let $t = |T_{f(u)}| \in [2^i, 2^{i+1})$ at the marginal moment of $(u, f(u))$. Note that at the marginal moment, u has another child $v \neq f(u)$ such that $|T_v| = t$, since $(u, f(u))$ becomes safe after the next operation.

Since $(u, f(u))$ can only be touched by $2^{i-1} \le \frac{t}{2}$ operations, at any moment when $(u, f(u)) \in C_i$ we have $\max_{y\in \text{Child}(u)}\{|T_y|\} \le 2\min\{|T_{f(u)}|, |T_v|\}$, as argued in Lemma 7. Hence for all joint $(u, f(u)) \in C_i$, we have $\max_{y\in \text{Child}(u)}\{|T_y|\} \le \frac{2}{3}|T_u|$. Let $(x_1, f(x_1)), (x_2, f(x_2)), \dots, (x_l, f(x_l))$ be all joints in C_i touched by this operation such that x_{j+1} is a descendant of x_j for all $j \in [l-1]$, we have

$$2^i - 2^{i-1} \le |T_{f(x_l)}| \le \frac{2}{3}|T_{x_l}| \le (\frac{2}{3})^{l-1}|T_{x_2}| \le (\frac{2}{3})^{l-1}(2^{i+1}+2^{i-1}) = 5(\frac{2}{3})^{l-1}2^{i-1},$$

which implies that $l \le 1 + \left\lfloor \frac{\log 5}{\log \frac{3}{2}} \right\rfloor = 4$. □

Lemma 9. *Alg. 4 keeps all joints maintained for any sequence of operations, if initially all joints are intact.*

Proof. Assume the contrary and let $(u, f(u))$ be the first joint that is not maintained such that all other joints in T_u are maintained. Consider the last moment t when $(u, f(u))$ is safe and the moment t' when $(u, f(u))$ becomes not maintained. Let $r = |T_{f(u)}|$ at moment t and $r' = |T_{f(u)}|$ at moment t'. Let $v \in \text{Child}(u)$ such that $|T_v| > 4r'$ at moment t'. Note that $|T_v| \le 2r$ at moment t since $(u, f(u))$ is safe at moment t. Let m be the number of operations that touch $(f(u), f(f(u)))$ or $(v, f(v))$ between moment t and t'. Similar to the proof of Lemma 7, we have $m \ge |r - r'| + |2r - 4r'| \ge \frac{r}{2}$.

Now consider all joints in $T_{f(u)}$ and T_v at moment t. Note that there are at most $3r$ joints. We count the number of reconstructions of risky joints between moment t and t'. A risky joint may become risky again after being rebuilt and hence a joint may be counted multiple times. Note that the classes do not contain the touched joints $(u, f(u))$ such that $|T_{f(u)}| = 1$. However, at most three such joints are touched by each operation. The total number of reconstructions of risky joints between moment t and t' can be upper bounded by $3r + 3m$+the number of joints that are removed from the node-classes, which by Lemma 8, is $3r + 3m + \sum_{i\in[\log n]} \frac{4m}{2^{i-1}} \le 3r + 3m + 8m \le 17m$.

Since by assumption $(u, f(u))$ is not maintained at moment t', by Alg. 4 line 5, $k_1 = 18$ risky joints in $T_{f(u)}$ or T_v are rebuilt after each of those m operations, which implies that $k_1 m = 18m \le 17m$ and is a contradiction. □

4.3 Time Complexity Analysis

Lemma 10. *The time complexity of Algorithm 4 is $O(\log n)$.*

Proof. As argued in Lemma 3, updating the Dynamic-Path-Shortcut after inserting or deleting one node can be done in $O(\log n)$ time. As attached in Appendix C, rebuilding a risky bucket/joint/node can be done in $O(\log n)$ time. Hence we only need to find the risky bucket/joints/nodes in $O(\log n)$ time.

If we use a max-heap to store the sizes of all buckets and a min-heap to store the sizes of the buckets that contain only a fraction of some centroid path, then it takes $O(\log n)$ time to find the bucket with maximum size and the bucket with minimum size such that does not contain a whole centroid path. Note that if there exist any risky buckets, then one of those two buckets must be risky. We prove the following fact in Appendix D.

Claim 2. *It takes $O(\log n)$ time to find the k_1 touched risky joints that are closest to x after each operation $(x, insertion/deletion)$.*

As proved in Lemma 1, at most $O(\log n)$ nodes can be touched by each operation and hence $O(\log n)$ time suffices to update their weights and to identify the risky nodes. □

References

[1] Noga Alon and Baruch Schieber. Optimal preprocessing for answering on-line product queries. Technical report, 1987.

[2] Amitabha Bagchi, Adam L. Buchsbaum, and Michael T. Goodrich. Biased skip lists. *Algorithmica*, 42(1):31–48, 2005.

[3] Samuel W Bent, Daniel D Sleator, and Robert E Tarjan. Biased search trees. *SIAM Journal on Computing*, 14(3):545–568, 1985.

[4] Hans L. Bodlaender, Gerard Tel, and Nicola Santoro. Trade-offs in non-reversing diameter. *Nord. J. Comput.*, 1(1):111–134, 1994.

[5] Bernard Chazelle. Computing on a free tree via complexity-preserving mappings. *Algorithmica*, 2:337–361, 1987.

[6] Mark De Berg, Marc Van Kreveld, Mark Overmars, and Otfried Cheong Schwarzkopf. Computational geometry: Algorithms and applications. In *Computational Geometry*. Springer Berlin Heidelberg, 2000.

[7] Michael Elkin and Shay Solomon. Optimal euclidean spanners: really short, thin and lanky. In *STOC*, pages 645–654, 2013.

[8] William Hesse. Directed graphs requiring large numbers of shortcuts. In *SODA*, pages 665–669, 2003.

[9] Sofya Raskhodnikova. Transitive-closure spanners: A survey. In *Property Testing*, pages 167–196, 2010.

[10] Daniel Dominic Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983.

[11] Shay Solomon. From hierarchical partitions to hierarchical covers: optimal fault-tolerant spanners for doubling metrics. In *STOC*, pages 363–372, 2014.

[12] Shay Solomon and Michael Elkin. Balancing degree, diameter and weight in euclidean spanners. In *ESA (1)*, pages 48–59, 2010.

[13] Robert Endre Tarjan. Efficiency of a good but not linear set union algorithm. *J. ACM*, 22(2):215–225, 1975.

[14] Mikkel Thorup. On shortcutting digraphs. In *Proceedings of the 18th International Workshop on Graph-Theoretic Concepts in Computer Science*, WG '92, pages 205–211, London, UK, UK, 1993. Springer-Verlag.

[15] Mikkel Thorup. Parallel shortcutting of rooted trees. *J. Algorithms*, 23(1):139–159, 1997.

[16] Marc J. van Kreveld and Mark H. Overmars. Union-copy structures and dynamic segment trees. *J. ACM*, 40(3):635–652, 1993.

[17] Andrew Chi-Chih Yao. Space-time tradeoff for answering range queries (extended abstract). In *STOC*, pages 128–136, 1982.

A Proofs in Section 3

A.1 Static Path Shortcut

Proof of **Lemma 1**: First, we can calculate t_k for all $k \in [q]$ in $O(q)$ time. For ease of notation, we set $t_k = 0$ for all $k \leq 0$ and $t_k = t_q$ for all $k \geq q+1$. Then we run Alg. 1 as SPS$(P, 1, q, t)$, which finds the position m such that divide the given range evenly in terms of weights, and do the same recursively on the two sub-paths.

By checking integer $l = 0, 1, 2, \ldots$ for the two ranges $[1+2^l-1, 1+2^{l+1}-1]$ and $[q-2^{l+1}+1, q-2^l+1]$ simultaneously, the smallest l required as in line 3 of Alg. 1 can be found in $O(\min\{\log m, \log(q-m)\})$ time. Given l, the range containing m is of width 2^l and hence binary search takes $O(l) = O(\min\{\log m, \log(q-m)\})$ time to locate the position m. Let $g(x)$ be the time complexity of Alg. 1 when $j - i + 1 = x$. Note that $g(x) = 0$ for all $x \leq 2$. Then we have $g(q) = g(m-2) + g(q-m) + O(\min\{\log m, \log(q-m)\})$, which by mathematical induction implies that $g(q) \leq c(q - \log q) = O(q)$, for some constant c. Thus the Static-Path-Shortcut S of path P can be constructed in $O(q)$ time.

It is easy to observe that there is at most one shortcut edge added to each node. Hence we have $\mathsf{deg}_S = O(1)$. Next we analyze the hop-distance $h_G(u_1, u_i)$ between u_1 and u_i.

For any $1 \leq a \leq b \leq q$, we call $[a, b] = \{a, a+1, \ldots, b\}$ a *range* if SPS(P, a, b, t) is called during the execution of SPS$(P, 1, q, t)$. By construction of the shortcut edges, given any two ranges R_1 and R_2 such that $R_1 \cap R_2 \neq \emptyset$, we have $R_1 \subseteq R_2$ or $R_2 \subseteq R_1$. Suppose $i \in [a_2, b_2] \subseteq [a_1, b_1]$, then we have $\sum_{j=a_2}^{b_2} w_{u_j} \leq \frac{1}{2} \sum_{j=a_1}^{b_1} w_{u_j}$, by the definition of Static-Path-Shortcut. Hence we know that the number of ranges that contain i is at most $\log \frac{w(P)}{w_{u_i}}$, since the total weight doubles in each outer range. Let $[a_1, b_1], [a_2, b_2], \ldots, [a_k, b_k]$ be the whole sequence of ranges that contain i: $1 = a_1 < a_2 < \ldots < a_k \leq i \leq b_k \leq \ldots \leq b_2 \leq b_1 = q$. Note that for each $1 \leq j < k$ there exists a straight path from u_{a_j} to $u_{a_{j+1}}$ of size at most 2. Since $[a_k, b_k]$ is the minimum width range that contains i, then either $a_k = b_k$ or we have one of the following: (1) $i = a_k$; (2) u_i is connected to u_{a_k} by a shortcut edge. Hence we have a straight path from $u_1 = u_{a_1}$ to u_i of size at most $2 \log \frac{w(P)}{w_{u_i}} + 1 = O(\log \frac{w(P)}{w_{u_i}})$, which implies that $h_G(u_1, u_i) = O(\log \frac{w(P)}{w_{u_i}})$. □

A.2 Dynamic Path Shortcut

Proof of **Lemma 2**: It is easy to observe by construction and by Lemma 1 that there is at most a constant number of shortcut edges added to each node. Hence we have $\mathsf{deg}_S = O(1)$.

By Definition 11, we can construct a straight path from u_1 to each u_i as follows. If $u_i \in B_1$, then by Lemma 1, we already have $h_G(u_1, u_i) = O(\log \frac{w(P)}{w_{u_i}})$ by following the Static-Path-Shortcut. Otherwise, let B_{x+1} be the bucket containing u_i. Starting from u_1, we can reach $u_{b_1-h_1+1}$ by one shortcut edge. Since there is a shortcut edge between any two consecutive buckets in each t-list in the biased-skip-list, we can reach $u_{b_x-h_x+1}$ from $u_{b_1-h_1+1}$ following Direct_Search(h, s, x)

using $O(\log \frac{w(P)}{w(B_x)})$ edges, by Fact 1. Then via shortcut edge $(u_{b_x-h_x+1}, u_{b_x})$ and path edge (u_{b_x}, u_{b_x+1}), we can reach the first node in bucket B_{x+1}. By Lemma 1, we can reach $u_i \in B_{x+1}$ using $O(\log \frac{\sum_{u\in B_{x+1}} w_u}{w_{u_i}}) = O(\log \frac{w(B_x)}{w_{u_i}})$ edges. Hence we have $h_G(u_1, u_i) = O(\log \frac{w(P)}{w(B_x)}) + O(\log \frac{w(B_x)}{w_{u_i}}) + O(1) = O(\log \frac{w(P)}{w_{u_i}})$.

Similar to the above analysis, we can use finger search to construct a straight path from u_i to u_j. W.l.o.g, assume that $i < j$. Note that $h_G(u_i, u_j) = O(\log(4n))$ is trivial if u_i and u_j are contained in a constant number of consecutive buckets. Otherwise let $u_i \in B_{x-1}$ and $u_j \in B_{y+1}$. Note that we can reach $u_{b_x-h_x+1}$ from u_i and reach u_j from $u_{b_y-h_y+l}$ via $O(\log(4n))$ edges, for any $l \in [h_y]$. Hence following Finger_Search(h, s, x, y), we can reach $u_{b_y-h_y+l}$ from $u_{b_x-h_x+1}$, for some $l \in [h_y]$ using $O(\log w(P))$ edges by Fact 1, which implies that $h_G(u_i, u_j) = O(\log(4n))$. □

Proof of **Lemma 3**: During the following analysis, as long as the size of a bucket is changed, we apply the following update to the bucket. If the bucket is of size less than $4\log(4n)$, then it is merged with an arbitrary neighbour bucket (if there is any) in the biased-skip-list. If the bucket is of size larger than $8\log(4n)$, then it is evenly divided into two buckets. Note that we not only maintain a proper Path-Decomposition, but further restrict the bucket size to be within $[4\log(4n), 8\log(4n)]$, which is stronger than definition of buckets.

If we add, remove or re-weight one node $u \in P$, then we can reconstruct one bucket such that u is added, removed or re-weighted in the new bucket. Note that by Fact 1, the biased-skip-list of the buckets can be maintained in $O(\log(4n))$ time, since only the weight of the bucket before the new bucket need to be changed. By Definition 11 and Lemma 1, only $O(\log(4n))$ shortcut edges related to the new bucket need to be rebuilt. Hence, the Dynamic-Path-Shortcut can be maintained in $O(\log(4n))$ time.

Suppose edge (u, v) is deleted and the path is split into two paths. If u and v are contained in two buckets, then we can directly split the biased-skip-list into two; otherwise we divide the bucket that contains u and v into two buckets and then split the biased-skip-list into two to separate those two buckets. Note that the above procedure only add, delete or re-weight a constant number of buckets while splitting the biased-skip-list. Fact 1 and Lemma 1 implies that the Dynamic-Path-Shortcut can be maintained in $O(\log(4n))$ time.

Similar to the above analysis, to concatenate a path P' to the end of path P, it suffices to add, remove or re-weight a constant number of boundary buckets while merging two biased-skip-lists into one. Hence by Fact 1, the Dynamic-Path-Shortcut can also be maintained in $O(\log(4n))$ time. □

B Maintaining the Nodes

The analysis for maintaining the nodes is quite similar to the above analysis. We present a brief proof in this section. The status of a node u will only be changed when a node is inserted to or deleted from T_v for some $v \in$ Child(u)

and $v \neq f(u)$. For each node u, between its two consecutive rebuilds, we call the last moment before u is changed from intact to safe the *marginal moment* of u.

Definition 14 (Node-Class). *We put nodes into* node-classes C_i, $i \in [\log n]$ *right after their marginal moments. Node u is put into C_i iff $\hat{w}_u \in [2^i, 2^{i+1})$. Node $u \in C_i$ is removed if u is touched by 2^{i-1} operations after its marginal moment.*

Lemma 11. *For all $i \in [\log n]$, each node $u \in C_i$ is either safe or intact.*

Proof. Let $t = \hat{w}_u$ at the marginal moment of u. Note that $w_u = t \geq 2^i$ at this moment.

Consider the time when u becomes risky and let $t' = \hat{w}_u$ at this moment. Then we have $t = w_u \geq 2t'$ or $t = w_u \leq \frac{t'}{2}$ at this moment. Hence the number of operations that touches u before u becomes risky is at least $|t - t'| \geq \frac{t}{2} \geq 2^{i-1}$. □

Lemma 12. *Each operation touches at most* 16 *nodes in each node-class.*

Proof. Fix one operation and one node-class C_i. Let $x_1, \ldots, x_l$ be all touched nodes in C_i such that x_{j+1} is a descendant of x_j for all $j \in [l-1]$. By Lemma 9, all joints are maintained and hence $|T_{f(x_j)}| \geq \frac{1}{4}|T_{x_{j+1}}| \geq \frac{1}{4}\hat{w}_{x_{j+1}}$ for all $j \in [l-1]$. Hence for all $j \in [l-2]$, $\hat{w}_{x_j} \geq \hat{w}_{x_{j+1}} + |T_{f(x_{j+1})}| \geq \hat{w}_{x_{j+2}} + \frac{1}{4}\hat{w}_{x_{j+2}} = \frac{5}{4}\hat{w}_{x_{j+2}}$. Since $\hat{w}_{x_j} \in [2^{i-1}, 5 \cdot 2^{i-1})$ for all $j \in [l]$, we have $5 \cdot 2^{i-1} \geq \hat{w}_{x_1} \geq \frac{5}{4}\hat{w}_{x_3} \geq (\frac{5}{4})^{\lfloor \frac{l-1}{2} \rfloor}\hat{w}_{x_l} \geq (\frac{5}{4})^{\lfloor \frac{l-1}{2} \rfloor} 2^{i-1}$, which implies that $\lfloor \frac{l-1}{2} \rfloor \leq \left\lfloor \frac{\log 5}{\log \frac{5}{4}} \right\rfloor = 7$ and $l \leq 16$. □

Lemma 13. *Alg. 4 keeps all nodes maintained for any sequence of operations, if initially all nodes are intact.*

Proof. Assume the contrary and let u be the first node that is not maintained such that all other nodes in T_u are maintained. Consider the last moment t when u is safe and the moment t' when u becomes not maintained. Let $r = \hat{w}_u$ at moment t and $r' = \hat{w}_u$ at moment t'. Hence $\frac{1}{2}r \leq w_u \leq \frac{1}{4}r'$ or $4r' \leq w_u \leq 2r$, which means that the operations that touch u between moment t and t' is $m \geq |r - r'| \geq \frac{r}{2}$.

There are at most r nodes in $T_u \setminus T_{f(u)}$ at moment t. Note that the node-classes do not contain the touched nodes u such that $\hat{w}_u = 1$. However, at most one such node is touched by each operation. The total number of reconstructions of risky nodes between moment t and t' can be upper bounded by $r + m +$the number of nodes that are removed from the classes, which by Lemma 12, is $r + m + \sum_{i \in [\log n]} \frac{16m}{2^{i-1}} \leq r + m + 32m \leq 35m$. Since by assumption u is not maintained at moment t', by Alg. 4 line 6, $k_2 = 36$ risky nodes other than u in T_u are rebuilt after each of those m operations, which implies that $k_2 m = 36m \leq 35m$ and is a contradiction. □

C Reconstruction Algorithms

The following algorithms can be used to rebuild a risky object and turn it intact. To assist the reconstruction, we can store and maintain $\hat{w}_u$ for each node $u \in V$ and $|T_u|$ for the top-node u of each centroid path. Note that by Claim 1, only $O(\log n)$ number of those values need to be updated after each operation. By Fact 1 and Lemma 3, the complexity of the following three update algorithms is $O(\log n)$.

Algorithm 5 Rebuild_Bucket(B, P):

```
Input: Bucket $B$ in the Path-Decomposition of path $P$
1: if $|B| < 4\log(4n)$ then
2:     merge $B$ with an arbitrary neighbor bucket
3: if $|B| > 8\log(4n)$ then
4:     split $B$ evenly into two buckets
5: update related bucket weights
6: update the biased-skip-list on the buckets
7: update the dynamic-path-shortcut of $P$
```

Algorithm 6 Rebuild_Node(u):

```
1: $w_u \leftarrow \hat{w}_u = |T_u| - |T_{f(u)}|$
2: update the weight of the bucket whose weight depends on $w_u$
3: update the biased-skip-list on the buckets
4: update the dynamic-path-shortcut of the path containing $u$
```

Algorithm 7 Rebuild_Joint(u):

```
1: remove $(u, f(u))$ and split the centroid path containing $u$ into two centroid paths
2: $f(u) \leftarrow \arg\max_{v \in \mathrm{Child}(u)} \{|T_v|\}$
3: add $(u, f(u))$ and merge the centroid paths containing $u$ and $v = f(u)$
4: Rebuild_Node($u$)
5: Rebuild_Bucket($B$) for all changed buckets $B$
```

D Proof of Claim 2

We use a modified version of dynamic segment tree [16], [6], with the *slackness* of a node as key, to find risky joints in each centroid path.

Definition 15 (Dynamic-Segment-Tree). *Given a centroid path $P = (u_1, u_2, \ldots, u_q)$ and a biased-skip-list h (with the successor pointer s) on the nodes, the dynamic-segment-tree of P is defined as follows. The node set of the dynamic-segment-tree* *is* $V = \{v_{i,j} | i = j \text{ or } \exists k \leq h_{max}, s_{i,k} = j+1\} \cup \{v_{1,q}\}$*, among which node* $v_{1,q}$ *is the root. We use* $p(v_{i,j})$ *to denote the parent of node* $v_{i,j} \in V$*. For all non-root node* $v_{i,j}$*, let* $p(v_{i,j}) = v_{k,l}$ *iff (1)* $[i,j] \subsetneq [k,l]$*; (2) there does not exist any* $v_{a,b} \in V$ *such that* $[i,j] \subsetneq [a,b] \subsetneq [k,l]$*.*

By properties of biased-skip-list [2], the dynamic-segment-tree we constructed guarantees that the height of the tree is $O(\log n)$ and each internal node in V has a constant number of children. By considering each $v_{i,j} \in V$ as an integer interval $[i,j] = \{i, i+1, \ldots, j\}$, we use $v_{i,j} \subseteq [k,l]$ to denote that $[i,j] \subseteq [k,l]$.

Definition 16 (Slackness). *The* slackness $S(v_{i,j})$ *of node* $v_{i,j} \in V$ *is defined as follows.*

- *If* $i = j$*, then* $S(v_{i,j}) = |T_{f(u_i)}| - \lceil \frac{1}{2} \max_{x \in Child(u_i) \setminus \{f(u_i)\}} \{|T_x|\} \rceil$ *if* $|Child(u_i)| \geq 2$ *and* $S(v_{i,j}) = \infty$ *otherwise.*
- *If* $i \neq j$*, then* $S(v_{i,j}) = \min_{x \in [i,j]} \{S(v_{x,x})\}$*.*

By definition, joint $(u_i, f(u_i))$ in P is risky iff $S(v_{i,i}) < 0$. Recall that a joint $(u_i, f(u_i))$ is touched by operation $(x, \text{insertion/deletion})$ if x is a descendant of u_i. If x is in $T_{f(u_i)}$, then we should increase or decrease $S(v_{i,i})$ by 1; otherwise, we need to recompute $S(v_{i,i})$.

Definition 17 (Slackness-Difference). *The* slackness-difference *of each* $v_{i,j} \in V$ *is defined as follows. Let* $d(v_{i,j}) = S(v_{i,j})$ *if* $v_{i,j} = v_{1,q}$ *and* $d(v_{i,j}) = S(v_{i,j}) - S(p(v_{i,j}))$ *otherwise.*

The slackness-difference $d(v_{i,j})$ is attached to each node $v_{i,j} \in V$ and maintained throughout the sequence of operations. Although the slackness is not stored, the following fact implies that with the help of $d(v_{i,j})$, the slackness of each node can be easily calculated.

Fact 2. *For all* $v_{i,j} \in V$*, let* $\hat{P}$ *be the path from* $v_{i,j}$ *to* $v_{1,q}$ *in the dynamic-segment-tree, then we have* $S(v_{i,j}) = \sum_{v_{k,l} \in \hat{P}} d(v_{k,l})$*.*

By the Fact 2, if we increase (or decrease) $d(v_{i,j})$ by t, then the slackness of each descendant of $v_{i,j}$ will be increased (or decreased) by t. Moreover, if $d(v_{i,j})$ is changed and $v_{k,l}$ is an ancestor of $v_{i,j}$, then $d(v_{k,l})$ and $d(v_{a,b})$ need to be changed accordingly, where $v_{a,b} \in \text{Child}(v_{k,l})$, since $v_{k,l}$ may have descendant whose slackness is not changed.

Lemma 14. *The dynamic-segment-trees together with the slackness-difference on each node of all centroid paths in* $T(V,E)$ *can be maintained in* $O(\log n)$ *time after each operation.*

Proof. Given a path P with weight attached to each node, by the construction and the maintainance of biased-skip-list [2], it can be verified that the dynamic-segment-tree(s) together with the slackness-difference attached to each node can be maintained in $O(\log n)$ time for the following operations: (1) inserting, deleting or re-weighting a node $u \in P$; (2) splitting the path P into two paths by deleting one edge in P; (3) concatenate a path P', who has its own dynamic-segment-tree, to the end of path P.

Notice that after each operation on the tree, the dynamic-path-shortcut will be updated as described in Section 3 and a constant number of risky buckets, joints and nodes will be rebuilt. Hence, the dynamic-segment-tree of each centroid path can be updated accordingly as described above. Besides the above updates, if some joints in centroid path P are touched after one operation, then the slackness-differences of nodes in the dynamic-segment-tree of P need to be updated, since the slackness of some nodes $v_{i,i}$ are changed.

Fix any centroid path $P = (u_1, u_2, \ldots, u_q)$. Let $\{u_1, u_2, \ldots, u_x\}$ be the whole set of nodes such that for all $i \in [x]$, $(u_i, f(u_i))$ is touched after one operation. By definition of slackness, we know that (i) for all $v_{i,j} \subseteq [1, x-1]$, $S(v_{i,j})$ need to be increased or decreased by 1 and the slackness-differences of the ancestors of $v_{i,j}$ need to be updated accordingly; (ii) $S(v_{x,x})$ need to be recalculated and the slackness-differences of the ancestors of $v_{x,x}$ need to be updated accordingly. Let $y \in \text{Child}(x)\backslash\{f(x)\}$ be the node touched by the same operation. We show that update (i) and (ii) can be done in $O(\log \frac{w(P)}{|T_y|})$ time.

By Fact 1, we can find the path P^* in the dynamic-segment-tree of P from $v_{x,x}$ to $v_{1,q}$ in $O(\log \frac{w(P)}{w_x})$ time, which means that $|P^*| = O(\log \frac{w(P)}{w_x})$. Let $Q = \{v_{i,j} | v_{i,j} \in P^* \text{ or } p(v_{i,j}) \in P^*\}$. Then we also have $|Q| = O(\log \frac{w(P)}{w_x})$.

For update-(i), let $T = \{v_{i,j} | v_{i,j} \subseteq [1, x-1] \text{ and } p(v_{i,j}) \not\subseteq [1, x-1]\}$ and $\tilde{T} = \{v_{i,j} | v_{i,j} \in T \text{ or } \exists v_{k,l} \in T, v_{i,j} \text{ is an ancestor of } v_{k,l}\}$. Then we only need to update $d(v_{i,j})$ for all $v_{i,j} \in \tilde{T}$, since for all $v_{k,l} \subseteq [1, x-1]$, there exists exactly one ancestor of $v_{k,l}$ in T. Note that $\tilde{T} \subseteq Q$ since for all $v_{i,j} \in T$, all ancestors of $v_{i,j}$ are contained in P^*. Hence $|\tilde{T}| = O(\log \frac{w(P)}{w_x})$ and update-(i) can be done in $O(\log \frac{w(P)}{w_x}) = O(\log \frac{w(P)}{|T_y|})$ time.

For update-(ii), based on the slackness-differences of all nodes in Q, we can calculate the slackness (before the operation) of all nodes in Q in $O(\log \frac{w(P)}{w_x})$ time. To update the slackness of $v_{x,x}$, we only need to find out whether $|T_y| = \max_{z \in \text{Child}(x)\backslash\{f(x)\}} |T_z|$. Note that this can be done in $O(\log \frac{w_x}{|T_y|})$ time by maintaining a biased-search-tree [3] on $\text{Child}(x)\backslash\{f(x)\}$, with the tree-size as key and weight. Given the recalculated value of $S(v_{x,x})$ and all other slackness of nodes in Q, we can update their slackness in $|Q| = O(\log \frac{w(P)}{w_x})$ time. Hence update-(ii) can be done in $O(\log \frac{w(P)}{w_x}) + O(\log \frac{w_x}{|T_y|}) = O(\log \frac{w(P)}{|T_y|})$ time.

Note that y must be the top-node of some centroid path P'. Hence we have $|T_y| \geq \frac{1}{4} w(P')$. By using a similar argument as in Lemma 4, we know that the total update time for all dynamic-segment-trees (with the slackness-differences) of the centroid paths is $O(\log n)$. □

Proof of **Claim 2**: Assume that all slackness-differences are maintained, we show how to determine the k_1 touched risky joints that are closest to node v, after each operation (v, insertion/deletion). Since k_1 is constant and we can rebuild a risky joint in $O(\log n)$ time once it is found, we only need to show how to find the risky joint that is closest to v.

By Claim 1, the touched joints are contained in $O(\log n)$ centroid paths. By checking whether the slackness-difference of the root node of the dynamic-segment-tree of each centroid path is negative or not, we can identify in $O(\log n)$ time the centroid path closest to v that contains a risky joint. Let P be that centroid path and $\{u_1, u_2, \ldots, u_x\}$ be the whole set of nodes such that $(u_i, f(u_i))$ is touched. We use the same definition of T and $\tilde{T}$ as above, where $|\tilde{T}| = O(\log \frac{w(P)}{w_x}) = O(\log n)$. As argued above, we can calculate the slackness of each node in $\tilde{T}$ in $O(\log n)$ time. Then in $O(\log n)$ time, we can find out the node $v_{i,j}$ in T with negative slackness that is closest to x. Starting from $v_{i,j}$, we can find out the child of $v_{i,j}$ with negative slackness that is closest to x in $O(1)$ time. By doing this recursively for $O(\log n)$ rounds, we can reach the leaf node $v_{k,k}$ with negative slackness that is closest to x, which implies that $(u_k, f(u_k))$ is the risky joint that is closest to x.

Hence the risky joint that is closest to v, after each operation $(v, \text{insertion/deletion})$, can be found in $O(\log n)$ time, which finishes the proof. □