# Top-Down Analysis of Path Compression[*]

Raimund Seidel[†] Micha Sharir[‡]

April 25, 2003

**Abstract**

We present a new analysis of the worst case cost of path compression which is an operation that is used in various well-known "Union-Find" algorithms. In contrast to previous analyses which are essentially based on bottom-up approaches our method proceeds top-down yielding recurrence relations from which the various bounds arise naturally. In particular the famous quasi-linear bound involving the inverse Ackermann function can be derived without having to introduce the Ackermann function itself.

## 1 Introduction

Path compression is used in a number of algorithms, most notably in various very natural solutions to the so-called Union-Find problem. This problem is basic enough to be covered in most introductory texts on algorithms, however the performance analysis of the solutions is more often than not at best incomplete if not omitted altogether. Already the definition of the function $\alpha$, the interesting constituent of the time bound, as a quasi inverse of the Ackermann function is complicated.

Let us briefly recall the Union-Find problem. It asks to maintain a partition of a finite set $X$ and a representative element $\rho(A) \in A$ for each

[*]Work on this paper has been supported by a grant from the U.S.–Israel Binational Science Foundation, by NSF Grants CCR-97-32101 and CCR-00-98246. by a grant from the Israeli Academy of Sciences for a Center of Excellence in Geometric Computing at Tel Aviv University, and by the Hermann Minkowski–MINERVA Center for Geometry at Tel Aviv University.

[†]Fachrichtung Informatik, Univ. des Saarlandes, Postfach 151150, D-66041 Saarbrücken, GERMANY. rseidel@cs.uni-sb.de

[‡]Tel Aviv Univ., School of Computer Science, Tel Aviv 69978, ISRAEL, and Courant Institute of Mathematical Sciences, New York Univ., New York, NY 10012, USA. michas@post.tau.ac.il

set $A$ participating in the partition. There are two operations: For $x \in X$ the query $\text{FIND}(x)$ is to determine the representative element $\rho(A_x)$, where $A_x$ is the set in the partition that contains $x$. For two sets $A$ and $B$ in the current partition the operation $\text{UNION}(\rho(A), \rho(B))$ is to change the partition, replacing $A$ and $B$ by their union and providing the new representative element $\rho(A \cup B)$.

A very natural solution of this Union-Find problem is to represent the partition as a forest of rooted trees on the node set $X$, where each tree represents a set of the partition and its root is the representative element of that set. The operation $\text{FIND}(x)$ is realized by traversing the path from $x$ to the root $\rho$ of its containing tree (just follow parent pointers) and reporting $\rho$. The operation $\text{UNION}(\rho(A), \rho(B))$ is realized by making $\rho(A)$ the parent of $\rho(B)$, or vice versa, thus combining the two trees. Note that a single parent pointer per node suffices to implement such forests.

The time thus necessary for a UNION-operation is constant, whereas for a FIND-operation it is proportional to the length of the path traversed. Thus it is advantageous to keep paths short. To this end it makes sense to be judicious in the UNION-operation which of the two involved roots to make the new root. Choosing the one whose tree has the larger number of nodes is known as the *linking-by-weight* strategy; choosing the one with larger *rank* is known as the *linking-by-rank* strategy. Here "rank" is an integer associated with node $x$ that is initially 0 and that is increased by one whenever $x$ is made a parent to a node of equal rank.[1] In both strategies ties are broken arbitrarily.

Both strategies produce trees of at most logarithmic height. Thus a sequence of UNION and $m$ FIND operations takes at most $O(n + m \log n)$ time. Here $n = |X|$ and the initial partition consists of all singleton subsets of $X$. Note that at most $n - 1$ UNION operations can occur.

Another way of producing short find-paths is so-called *path compression*: When performing a FIND operation make all nodes of the traversed path children of the root. This increases the running time of the FIND operation only by a constant factor but may make the find-paths of subsequent operations shorter.

Analyzing the performance of algorithms employing path compression in combination with various linking strategies has a long history.

Hopcroft and Ullman [3] proved a bound of $O(m \log^* n)$. Their approach was refined by Tarjan [5] to the so-called "multiple partition method" leading

[1] In other words the rank of $x$ is nothing but the height (number of edges on longest leaf-root path) of the tree rooted at $x$ *if no path compression had occurred.*

to a bound of $O(m\alpha_T(m,n))$, where

$$\alpha_T(m,n) = \min\{i \geq 1 \mid A(i, \lfloor m/n \rfloor) > \lceil \log_2 n \rceil\}$$

and $A$ is the "Ackermann function" defined by

$$\begin{array}{lcll} A(1,j) & = & 2^j & \text{for } j \geq 1, \\ A(i,1) & = & A(i-1,2) & \text{for } i \geq 2, \\ A(i,j) & = & A(i-1, A(i,j-1)) & \text{for } i,j \geq 2. \end{array}$$

Tarjan [6] also showed that this slightly superlinear bound was best possible for a reasonable, pointer based model of computation. Kozen [4] simplified Tarjan's analysis of the upper bound. Later Tarjan [8] and also Harfst and Reingold [2] cast the existing analyses in terms of potential functions, a standard tool for the amortized analysis of algorithm performance. This type of analysis was taken into [1].

All the analyses so far have proceeded by low-level accounting and charging individual parent pointer changes to operations or nodes and bounding the grand total of those charges. In this paper we present an analysis that is based on a radically different approach. We proceed top-down and employ divide-and-conquer to derive recurrence relations from which the bounds follow. Our analysis yields rather precise bounds. For example, if $n = |X| < 2^{66}$ and linking-by-rank and path compression are used, then any sequence of $m$ Find operations causes at most $\min\{m+4n, 2m+2n\}$ parent pointer changes.

## 2 The main lemma

Let $\mathcal{F}$ be a forest of disjoint rooted trees on a finite node set $X$. Here we only consider paths $p$ in $\mathcal{F}$ that lead from some node $x$ to some ancestor $y$ of $x$. If $y$ is a root in $\mathcal{F}$ we refer to $p$ as a *rootpath*, otherwise we call it a *non-rootpath* and we denote the parent of $y$ by $a(p)$.

*Compressing a non-rootpath* $p$ means minimally changing $\mathcal{F}$ so that every node on $p$ has $a(p)$ as its parent. *Compressing a rootpath* $p$ means minimally changing $\mathcal{F}$ so that every node on $p$ becomes a root. We define the cost of such an operation to be the number of nodes that get a new parent, i.e. if $p$ is a non-rootpath with $d$ nodes the cost is $d-1$, and if $p$ is a rootpath the cost is 0. It is also convenient to allow empty paths involving no nodes. We classify them as rootpaths and "compressing" them has cost 0.

Let $C = (p^{(i)})_{1 \leq i \leq M}$ be a sequence of paths along which compressions are performed starting with the initial forest $\mathcal{F}$, i.e. $\mathcal{F}_0 = \mathcal{F}$ and for $1 \leq i \leq M$

the path $p_i$ is in $\mathcal{F}_{i-1}$ and $\mathcal{F}_i$ is obtained from $\mathcal{F}_{i-1}$ by compressing $p_i$. Let *cost*$(C)$ denote the cumulative cost of $C$, i.e. the total number of times that some node gets a new parent.

The notion of path compression that we just defined is more general than the path compressions that arise in FIND-operations. For proof purposes this generality turns out to be useful. Moreover, as already noted in [3] and [5], the following holds:

**Lemma 1** *Let $S$ be some sequence of* UNION *and $m$* FIND*-operations on an initial partition of an $n$-element set $X$ into singletons. Let $T$ be the time necessary to execute $S$.*

*There is a forest $\mathcal{F}$ on $X$ and a sequence $C$ of $m$ path compressions, all involving non-rootpaths, so that $T = O(m + n + cost(C))$.*

*Proof:* (Sketch) For $\mathcal{F}$ consider the forest that is generated by performing just the UNION-operations of the sequence. The sequence of FIND-operations then defines a sequence of non-rootpaths in the corresponding forest. □

Thus we are interested in upper bounds for *cost*$(C)$ measured in terms of $|X|$ and $|C|$, the number of non-rootpath compressions in $C$.

Let us create a setup for divide-and-conquer. Consider a partition of the node set $X$ of $\mathcal{F}$ into a "bottom set" $X_b$ and a "top set" $X_t$. We call the pair $(X_b, X_t)$ a *dissection for $\mathcal{F}$*, iff $X_t$ is is upwards closed in $\mathcal{F}$, i.e. if some node $x$ is in $X_t$, then every ancestor of $x$ in $\mathcal{F}$ must be in $X_t$ also. Note that a dissection $(X_b, X_t)$ cuts every path into two contiguous subpaths $p_b$ and $p_t$ consisting of the nodes on $p$ that are in $X_b$ and $X_t$, respectively. Either subpath maybe empty. Also note that upward-closedness and hence also dissections are preserved under path compression. Let $\mathcal{F}(X_b)$ and $\mathcal{F}(X_t)$ be the subforests of $\mathcal{F}$ induced by the sets $X_b$ and $X_t$.
All our analyses hinge on the following **Main Lemma**:

**Lemma 2** *Let $C$ be a sequence of $|C|$ compress operations in a forest $\mathcal{F}$ with node set $X$. Let $(X_b, X_t)$ be an arbitrary dissection for $\mathcal{F}$.*

*There are sequences $C_b$ and $C_t$ of compress operations for $\mathcal{F}(X_b)$ and $\mathcal{F}(X_t)$, respectively, with $|C_b| + |C_t| \leq |C|$ and*

$$cost(C) \leq cost(C_b) + cost(C_t) + |X_b| + |C_t|\,.$$

*Proof:* Let $C = (p^{(i)})_{1 \leq i \leq M}$ be the sequence of paths to be compressed and let $(\mathcal{F}^{(i)})_{0 \leq i \leq M}$ be the resulting sequence of forests.

Let $(X_b, X_t)$ be a dissection of $\mathcal{F} = \mathcal{F}^{(0)}$ and let $\mathcal{F}_b = \mathcal{F}(X_b)$ and $\mathcal{F}_t = \mathcal{F}(X_t)$ be the induced bottom and top forests. Define path sequences $C_b = (p_b^{(i)})_{1 \le i \le M}$ and $C_t = (p_t^{(i)})_{1 \le i \le M}$. It is easy to check inductively that for each $1 \le i \le M$ the path $p_b^{(i)}$ occurs in $\mathcal{F}^{(i-1)}(X_b)$ and compressing it produces $\mathcal{F}^{(i)}(X_b)$. Thus $C_b$ is a compression sequence for $\mathcal{F}_b$. Analogously $C_t$ is a compression sequence for $\mathcal{F}_t$.

If $p^{(i)}$ is a non-rootpath in $\mathcal{F}^{(i-1)}$ then at most one of $p_b^{(i)}$ and $p_t^{(i)}$ is a non-rootpath in its respective forest. If $p^{(i)}$ is rootpath in $\mathcal{F}^{(i-1)}$ then both $p_b^{(i)}$ and $p_t^{(i)}$ are rootpaths. Thus $|C_b| + |C_t| \le |C|$.

If a node from $X_t$ gets a new parent (necessarily from $X_t$) when compressing path $p^{(i)}$, then exactly the same happens when compressing $p_t^{(i)}$. Thus the number of those types of parent changes is given by $cost(C_t)$. Likewise the number of times that a node from $X_b$ gets a new parent from $X_b$ is given by $cost(C_b)$.

A node from $X_b$ getting a new parent which is in $X_t$ can only happen when compressing a non-rootpath $p^{(i)}$ for which $p_b^{(i)}$ is a non-empty rootpath in the bottom forest. In this case all but the topmost node on $p_b^{(i)}$ get a parent from $X_t$ for the first time, which can happen to each node in $X_b$ at most once. The topmost node gets a new parent again from $X_t$ only if $p_t^{(i)}$ is non-empty which means $p_t^{(i)}$ is a non-rootpath (since $p^{(i)}$ is a non-rootpath). So this type of event can happen at most $|C_t|$ times. Since $X_t$ is upwards closed no node in $X_t$ can get a parent from $X_b$ and thus the total number of parent changes, i.e. $cost(C)$, is bounded by

$$cost(C_b) + cost(C_t) + |X_b| + |C_t| \,.$$

□

## 3 Arbitrary Linking

Let $f(m,n)$ denote the maximum possible cost for a sequence of $m$ compress operations in a forest of $n$ nodes. By Lemma 1, $f(m,n)$ yields an asymptotic upper bound for the running time of $m$ union-find operations in a universe of $n$ elements if path compression and *arbitrary* linking is used.

Using our main lemma it is straightforward to give a rough argument that $f(m,n) \le (m+n/2)\log_2 n$ must hold: For a given compression sequence $C$ of length $m$, choose a dissection $(X_b, X_t)$, where both $X_b$ and $X_t$ have size about $n/2$. Using induction, the inequality of the main lemma then yields

$$\begin{aligned}
cost(C) &\leq (|C_b| + n/4)\log_2(n/2) + (|C_t| + n/4)\log_2(n/2) + n/2 + |C_t| \\
&\leq (m + n/2)\log_2(n/2) + n/2 + m \\
&\leq (m + n/2)\log_2 n\,.
\end{aligned}$$

For large $m$ this bound can be improved to

$$f(m,n) \leq (2m + n)\log_{\lceil m/n\rceil+1} n\,,$$

which is achieved by setting $k = \lceil m/n\rceil + 1$ in the following claim:

**Claim 3** *For any integer $k > 1$ we have $f(m,n) \leq (m + (k-1)n)\lceil\log_k n\rceil$.*

*Proof:* The bound clearly holds if $n \leq k$ since each node can get a new parent at most $n-2$ times. So assume $n > k$ and let $C$ be some sequence of $m$ compress operations in a forest with $n$ nodes. Let $(X_b, X_t)$ be a dissection with $n_t = |X_t| = \lceil n/k\rceil$ and $n_b = |X_b|$. Let $C_t$ and $C_b$ be the compression sequences asserted in the main lemma and let $m_t$ and $m_b$ be their respective sizes. Then, using induction, the inequality of the main lemma implies

$$\begin{aligned}
cost(C) &\leq (m_b + (k-1)n_b)\underbrace{\lceil\log_k n_b\rceil}_{\leq\lceil\log_k n\rceil} + (m_t + (k-1)n_t)\underbrace{\lceil\log_k n_t\rceil}_{=\lceil\log_k n\rceil-1} + n_b + m_t \\
&= (m_t + m_b + (k-1)(n_t + n_b))\lceil\log_k n\rceil - m_t - (k-1)n_t + n_b + m_t \\
&\leq (m + (k-1)n)\lceil\log_k n\rceil\,,
\end{aligned}$$

where the last inequality follows from the fact that $n_b \leq (k-1)n_t$ by construction. $\square$

## 4 Linking by rank

For every node $x$ in a forest $\mathcal{F}$ define its rank $rank(x)$ to be the height of the subtree rooted at $x$, where the height of a one-node tree is 0. We call $\mathcal{F}$ a *rank balanced forest*, or simply *rank forest*, if for each node $x$ in $\mathcal{F}$ the following property holds: For each $i$ with $0 \leq i < rank(x)$ node $x$ has at least one child $y_i$ with $rank(y_i) = i$.

Rank forests arise in union-find algorithms that employ the linking-by-rank strategy. It is easy to prove by induction that in such a forest every node of rank $k$ must be the root of a subtree of size at least $2^k$. Thus the maximum rank that occurs is at most $\log_2 n$, where $n$ is the number of nodes. The following inheritance lemma is important for our purposes. Its straightforward proof is left to the reader.

**Lemma 4** *Let $\mathcal{F}$ be a rank forest with node set $X$ and maximum rank $r$. Let $s$ be some integer, let $X_{\leq s}$ be the set of nodes with rank at most $s$, and let $X_{>s}$ be the set of nodes with rank exceeding $s$. Then*

*(i)* $(X_{\leq s}, X_{>s})$ *is a dissection.*

*(ii)* $\mathcal{F}(X_{\leq s})$ *is a rank forest with maximum rank at most $s$.*

*(iii)* $\mathcal{F}(X_{>s})$ *is a rank forest with maximum rank at most $r-s-1$.*

*(iv)* $|X_{>s}| \leq |X|/2^{s+1}$.

Let $f(m,n,r)$ be the maximum cost for a sequence of $m$ path compressions in a rank forest with $n$ nodes and maximum rank $r$. By Lemma 1 $f(m,n,\lfloor \log_2 n \rfloor)$ yields an asymptotic upper bound for the running time of $m$ union-find operations in a universe of $n$ elements if path compression and linking-by-rank are used.

We need two definitions involving integer functions, the first of which is standard. Let $g : \mathbb{N} \to \mathbb{N}$ be a function with $g(n) < n$ for $n > 0$. Define the functions $g^* : \mathbb{N} \to \mathbb{N}$ and $g^\diamond : \mathbb{N} \to \mathbb{N}$ as follows:

$$g^*(r) = \begin{cases} 0 & \text{if } r \leq 1 \\ 1 + g^*(g(r)) & \text{if } r > 1, \end{cases}$$

$$g^\diamond(r) = \begin{cases} g(r) & \text{if } g(r) \leq 1 \\ 1 + g^\diamond(\lceil \log_2 g(r) \rceil) & \text{if } g(r) > 1. \end{cases}$$

Note that $g^\diamond$ is essentially $(\log \circ g)^*$.

With these definitions we can state and prove the following **Shifting Lemma**.

**Lemma 5** *Assume there is some $k \geq 0$ and some non-decreasing function function $g : \mathbb{N} \to \mathbb{N}$ with $g(r) < r$ for $r > 0$ so that*

$$f(m,n,r) \leq km + 2ng(r)$$

*holds for all $m, n, r$. Then the following bound also holds for all $m, n, r$:*

$$f(m,n,r) \leq (k+1)m + 2ng^\diamond(r)\,.$$

*Proof:* We proceed by induction on $r$. The statement clearly holds if $g(r) \leq 1$, since in these cases we have $g^\diamond(r) = g(r) = 0$.

So assume $r$ is such that $g(r) > 1$ and let $C$ be some compression sequence of length $m$ in a rank forest $\mathcal{F}$ with $n$ nodes and maximum rank $r$. Let $s = \lceil \log_2 g(r) \rceil$ and let $(X_{\leq s}, X_{>s})$ be the dissection described in Lemma 4. Note that $s < r$ since $g(r) < r$. Put $n_b = |X_{\leq s}|$ and $n_t = |X_{>s}|$. Let $C_b$ and $C_t$ be the compression sequences asserted in the main lemma and put $m_b = |C_b|$ and $m_t = |C_t|$. By the main lemma we have

$$cost(C) \leq cost(C_b) + cost(C_t) + n_b + m_t\,.$$

The bottom forest $\mathcal{F}(X_{\leq s})$ is a rank forest of maximum rank $s < r$. Using the inductive assumption we get

$$\begin{array}{rcl} cost(C_b) & \leq & (k+1)m_b + 2n_b g^\diamond(s) \\ & \leq & (k+1)m_b + 2n\,\underbrace{g^\diamond(\lceil \log_2 g(r) \rceil)}_{g^\diamond(r)-1} \\ & = & (k+1)m_b + 2ng^\diamond(r) - 2n\,. \end{array}$$

The top forest $\mathcal{F}(X_{>s})$ is a rank forest of maximum rank $r - s - 1$ with

$$n_t \leq n/2^{s+1} = n/2^{1+\lceil \log_2 g(r \rceil} \leq n/(2g(r))$$

nodes. Using the assumption of the lemma and the fact that $g$ is non-decreasing, we therefore get

$$cost(C_t) \leq km_t + 2n_t g(r - s - 1) \leq km_t + n\,.$$

Putting things together we get

$$\begin{array}{rcl} cost(C) & \leq & \Big((k+1)m_b + 2ng^\diamond(r) - 2n\Big) + \Big(km_t + n\Big) + n_b + m_t \\ & \leq & (k+1)(m_b + m_t) + 2ng^\diamond(r) \\ & = & (k+1)m + 2ng^\diamond(r)\,. \end{array}$$

Since this is true for any sequence $C$ of length $m$ in a rank forest with $n$ nodes and maximum rank $r$, we get the desired bound

$$f(m,n,r) \leq (k+1)m + 2ng^\diamond(r)\,.$$

□

The shifting lemma makes it possible to take a simple but loose bound for $f(m,n,r)$ and derive from it a whole sequence of valid bounds. From these bounds we then choose a particularly good one.

**Corollary 6** *Let the integer functions $J_k$ with $k \in \mathbb{N}$ be defined as*

$$J_0(r) = \lceil (r-1)/2 \rceil \quad \textit{and} \quad J_k(r) = J_{k-1}^{\diamond}(r) \textit{ for } k > 0.$$

*Then for each $k \in \mathbb{N}$ we have*

$$f(m,n,r) \leq km + 2nJ_k(r)\,.$$

*Proof:* It is easy to show by induction that for each $k$ the function $J_k$ is non-decreasing and it satisfies $J_k(r) < r$ for all $r > 0$.

Since a node in a rank forest of maximum rank $r$ has at most $r$ ancestors (one of which is its initial parent) it can get a new parent at most $r-1$ times. Therefore

$$f(m,n,r) \leq n \cdot (r-1) \leq 2nJ_0(r)$$

holds. For $k > 0$ the stated bound now follows by induction using the shifting lemma. □

The $J_k$'s are very slowly growing functions even for small $k$'s. The reader may check that $J_1(r) \leq 2$ and $J_2(r) \leq 1$ for $r \leq 65$. Since in a rank forest with $n$ nodes and of maximum rank $r$ we have $r \leq \lfloor \log_2 n \rfloor$ we therefore get for $n < 2^{66}$ a bound of

$$f(m,n,r) \leq \min\{m + 4n, 2m + 2n\}\,.$$

For general $m$ and $n$ we now want to choose from the infinitely many bounds provided by the $J_k$'s one that is particularly good. To this end define

$$\alpha_S(m,n) = \min\{k \in \mathbb{N} | J_k(\lfloor \log_2 n \rfloor) \leq 1 + m/n\}\,.$$

Using the corollary we then get the following:

**Theorem 7** *For all $m, n, r$ we have $f(m,n,r) \leq (\alpha_S(m,n) + 2)m + 2n$.*

Invoking Lemma 1 we immediately get:

**Theorem 8** *Performing a sequence of* Union*-operations and $m$* Find*-operations on a set of size $n$ using union-by-rank and path compression requires at most $O(n + m\alpha_S(m,n))$ time.*

The reader may wonder how the function $\alpha_S(m,n)$ defined here relates to Tarjan's initial definition of the inverse Ackermann function $\alpha_T(m,n)$. In analogy to the formalism used in this paper, $\alpha_T$ can be defined as follows:

$$\alpha_T(m,n) = \min\{k \geq 1 \mid T_k(\lfloor \log_2 n \rfloor) < m/n\},$$

where

$$T_1(r) = \lfloor \log_2 r \rfloor \quad \text{and} \quad T_k(r) = T_{k-1}^{\bullet}(r) \text{ for } k > 1,$$

and

$$g^{\bullet}(r) = \begin{cases} 0 & \text{if } r \leq 2 \ \ \text{(in contrast to 1 in the definition of } g^*\text{)} \\ 1 + g^{\bullet}(g(r)) & \text{if } r > 2. \end{cases}$$

One can see that the differences between $\alpha_S$ and $\alpha_T$ are minor and one can show that asymptotically they are equivalent.

## 5 Linking by weight

Besides linking-by-rank the method of linking-by-weight is another standard strategy in union-find algorithms. Applying our top-down approach to its analysis is a bit less straightforward than in the case of linking by rank that we just saw. The main technical issue is to create a setting where some analogue of Lemma 4 holds.

Let $\mathcal{F}$ be a forest and for each node $x$ in $\mathcal{F}$ let $T_x$ be the subtree rooted at $x$, let $h_x$ be the height of $T_x$, and let $w(x)$ be some positive integer weight function on the nodes $x$ of $\mathcal{F}$. For each node $x$ define $W(x) = \sum_{y \text{ node in } T_x} w(y)$. We call $\mathcal{F}$ a *weight balanced forest* iff for every node $x$ in $\mathcal{F}$ with parent $y$ we have $W(y) \geq 2W(x)$. We call such a forest of type $(\mu, h, W)$ iff

(i) $w(x) \geq 2^{\mu}$ for every leaf $x$,

(ii) $h_x \leq h$ for each node $x$,

(iii) $\sum_{y \text{ node in } \mathcal{F}} w(y) \leq W$.

Note that any forest created by linking by weight with $n$ initial nodes, each of weight 1, is a weight balanced forest of type $(0, \lfloor \log_2 n \rfloor, n)$.

**Lemma 9** *Let $\mathcal{F}$ be a weight balanced forest with node set $X$ and node weightfunction $w$ and let $\mathcal{F}$ be of type $(\mu, h, W)$. Let $s$ be some integer, let*

*$X_{\leq s}$ be the set of the nodes $x$ with $h_x \leq s$ and let $X_{>s}$ be the set of nodes with $h_x > s$. Define a new weight function*

$$w'(x) = \begin{cases} w(x) & \textit{if } x \in X_{\leq s} \\ w(x) + \sum_{y \in X_{\leq s} \textit{ child of } x} W(y) & \textit{if } x \in X_{>s}. \end{cases}$$

*The following hold:*

*(i)* $(X_{\leq s}, X_{>s})$ *is a dissection.*

*(ii)* $\mathcal{F}(X_{\leq s})$ *with weight function* $w'$ *is a weight balanced forest of type* $(\mu, s, W)$.

*(iii)* $\mathcal{F}(X_{>s})$ *with weight function* $w'$ *is a weight balanced forest of type* $(\mu + s + 1, h - s - 1, W)$.

*(iv)* $|X_{\leq s}| \leq |X| < 2W/2^{\mu}$ *and* $|X_{>s}| < 2W/2^{\mu+s+1}$.

*Proof:* Points (i) and (ii) are trivial. Point (iii) follows from the property $W(\textit{parent}(x)) \geq 2W(x)$ and from the fact that the definition of the new weights $w'()$ implies that for each node $x \in X_t$ the value $W'(x)$, defined with respect to $\mathcal{F}(X_{>s})$, coincides with $W(x)$ defined with respect to $\mathcal{F}$.

For point (iv) it suffices to show that the number of nodes in a weight balanced forest $\mathcal{F}$ of type $(\mu, h, W)$ is less than $2W/2^{\mu}$. For each $i \in \mathbb{N}$ let $N_i = \{x \text{ node in } \mathcal{F} | h_x = i\}$ and let $W_i = \sum_{x \in N_i} W(x)$. Note that $W_i \leq W$ for each $i$. Since for each $x \in N_i$ we have $W(x) \geq 2^{\mu+i}$ (a consequence of the property $W(\textit{parent}(x)) \geq 2W(x)$) we get

$$|N_i| \leq W_i/2^{\mu+i} \leq W/2^{\mu+i}.$$

Summing over all $i$ yields the desired bound on the number of nodes in $\mathcal{F}$.
□

Let $f(m, \mu, h, W)$ denote the maximum cost for a sequence of $m$ path compressions in a weight balanced forest of type $(\mu, h, W)$. Again a shifting lemma holds:

**Lemma 10** *Assume there is some $k \geq 0$ and some non-decreasing function function $g : \mathbb{N} \to \mathbb{N}$ with $g(h) < h$ for $h > 0$ so that*

$$f(m, \mu, h, W) \leq km + 4(W/2^{\mu})g(h)$$

*holds for all $m, \mu, h, W$. Then also the following bound holds for all $m, \mu, h, W$:*

$$f(m, \mu, h, W) \leq (k + 1)m + 4(W/2^{\mu})g^{\diamond}(h).$$

The proof proceeds in the same way as the proof of Lemma 5, using the dissection properties described in Lemma 9 instead of the ones described in Lemma 4, and with $h$ playing the role of $r$.

We can now conclude that

$$f(m, \mu, h, W) \leq km + 4(W/2^{\mu})J_k(h)$$

for every $k \in \mathbb{N}$.

As already noted, in forests as they arise in union-find algorithms with linking by weight we have $w(x) = 1$ for each of the $n$ nodes, and hence $\mu = 0$, $W = n$, and $h \leq \log_2 n$. Thus we get the following:

**Theorem 11** *A sequence of $m$ path compressions in a forest of $n$ nodes as it arises in union-find algorithms with linking by weight has cost at most*

$$(\alpha_S(m, n) + 4)m + 4n\,.$$

Again invoking Lemma 1 we immediately get:

**Theorem 12** *Performing a sequence of* UNION*-operations and $m$* FIND*-operations on a set of size $n$ using union-by-weight and path compression requires at most $O(n + m\alpha_S(m, n))$ time.*

## 6 Open problems

Can this top-down analysis method be made to work also for variants of path compression such as path compaction (see [7])? Can this top-down approach be used to prove lower bounds involving the inverse Ackermann function?

## References

[1] THOMAS H. CORMAN, CHARLES E. LEISERSON, RONALD L. RIVEST, AND CLIFF STEIN: **Introduction to Algorithms.** Second Edition, MIT Press (2001).

[2] GREGORY C. HARFST, EDWARD M. REINGOLD: A Potential-based Amortized Analysis of the Union-Find Data Structure. **ACM SIGACT News** 31(3) September 2000.

[3] JOHN E. HOPCROFT, JEFFREY D. ULLMAN: Set Merging Algorithms. *SIAM J. Comput.* 2(4): 294–303 (1973).

[4] Dexter C. Kozen: **The Design and Analysis of Algorithms.** Springer Texts and Monographs in Computer Science (1991).

[5] Robert E. Tarjan: Efficiency of a Good But Not Linear Set Union Algorithm. *JACM* 22(2): 215–225 (1975).

[6] Robert E. Tarjan: A Class of Algorithms which Require Nonlinear Time to Maintain Disjoint Sets. *JCSS* 18(2): 110–127 (1979).

[7] Robert E. Tarjan, Jan van Leeuwen: Worst-case Analysis of Set Union Algorithms. *JACM* 31(2): 245–281 (1984).

[8] Robert E. Tarjan: Princeton cs423 class notes (1999) `http://www.cs.princeton.edu/courses/archive/spring99/cs423/handouts.html`