Attributing Authorship of Revisioned Content

Luca de Alfaro[*]
Computer Science Dept.
University of California
Santa Cruz, CA 95064, USA
luca@ucsc.edu

Michael Shavlovsky
Computer Science Dept.
University of California
Santa Cruz, CA 95064, USA
mshavlov@ucsc.edu

ABSTRACT

A considerable portion of web content, from wikis to collaboratively edited documents, to code posted online, is revisioned. We consider the problem of attributing authorship to such revisioned content, and we develop scalable attribution algorithms that can be applied to very large bodies of revisioned content, such as the English Wikipedia.

Since content can be deleted, only to be later re-inserted, we introduce a notion of authorship that requires comparing each new revision with the entire set of past revisions. For each portion of content in the newest revision, we search the entire history for content matches that are statistically unlikely to occur spontaneously, thus denoting common origin. We use these matches to compute the earliest possible attribution of each word (or each token) of the new content. We show that this "earliest plausible attribution" can be computed efficiently via compact summaries of the past revision history. This leads to an algorithm that runs in time proportional to the sum of the size of the most recent revision, and the total amount of *change* (edit work) in the revision history. This amount of change is typically much smaller than the total size of all past revisions. The resulting algorithm can scale to very large repositories of revisioned content, as we show via experimental data over the English Wikipedia.

Categories and Subject Descriptors

H.5.3 [**Information Interfaces and Presentation**]: Group and Organization Interfaces—*Computer-supported cooperative work*; I.7.1 [**Document and Text Processing**]: Document and Text Editing—*Version control*

Keywords

Authorship; Wikipedia; revisioned content

1. INTRODUCTION

Versioned content is abundant on the Web. Wikipedia, and wikis, constitute the most prominent example, and they account for a large portion of total page-views. Blogs with multiple authors, and pages served by content-management systems, are another example in which the versioning is present, but not directly exposed to the viewer. Code is another prominent example of revisioned content, and one that is becoming common on the web, thanks to the success of sites like GitHub, where users can share their code repositories.

We study in this paper the problem of attributing revisioned content to its author, and more generally, to the revision where it was originally introduced. This problem is insteresting for several reasons. The Wikipedia Reuse Policy[1] requires people reusing Wikipedia material to either provide a link to the original article and revision history, or to cite the most prominent authors of the content. Furthermore, in the Wikipedia community it is felt that proper content attribution is an important way to acknowledge and reward contributors, and to foster participation and contributions from communities where authorship has been traditionally recognized and rewarded, such as the academic community [7, 12]. Tracking the authorship of Wikipedia content is also an important tool in assisting editors, and viewers, in determining the origin of assertions, and analyzing page evolution. In code, as in wikis, authorship tracking is useful to properly reward contributors. Furthermore, authorship tracking can be useful in determining the reason behind implementation choices. Several revisioning systems implement "blame" methods, which attribute every line to an author/revision, but this attribution is extremely crude and imprecise, as it cannot cope with blocks of code that are transposed from one location to another, or from one file to another — changes that are common when code is polished or refactored.

At first glance, the attribution problem for revisioned content seems trivial: surely we can simply compare each revision with the previous one, detect any new content, and attribute it to the revision's author. Unfortunately, things are not quite so simple. Content in revisioned systems is often deleted, only to be later introduced, and it is important to be able to trace the authorship to the first original introduction. In Wikipedia, the content of pages is frequently removed by vandals, and re-instated in subsequent revisions: this is illustrated in Figure 9, where the periodic dips in page size correspond to content deletions. One way to guard against such attacks is to check whether the most recent revision happens to coincide with one of the previous revisions, in which case, authorship is carried over from the previous revision. However, this ad-hoc remedy cannot cope with broader attacks. For instance, attackers could first use

[*] The authors are listed in alphabetical order.

Copyright is held by the International World Wide Web Conference Committee (IW3C2). IW3C2 reserves the right to provide a hyperlink to the author's site if the Material is used in electronic media.
WWW 2013, May 13–17, 2013, Rio de Janeiro, Brazil.
ACM 978-1-4503-2035-1/13/05.

[1] http://en.wikipedia.org/wiki/Wikipedia:Reusing_Wikipedia_content

a fake identity to remove the page contents, then use their main identity to restore the page to its previous contents, except for some small, imperceptible changes that foil the revision equality check: the whole page content would then be attributed to them. The goal of this work is to present algorithms that can be used on Wikipedia, with the resulting authorship information available to visitors. Once authorship information is prominently displayed, attacks that aim at inflating the size of one's authorship are likely, prompting our quest for general, robust algorithms. A more general solution also benefits code attribution, since blocks of code are commonly moved from one branch to another, or deleted and later re-inserted.

We propose to attribute authorship of revisioned content by comparing the content of the most recent revision, with the entire content of all previous revisions. For every symbol (word, or character, or token) in the most recent revision, we compute all statistically significant matches with previous content: these are the matches whose sequence of symbols is rare enough that the match is likely to be due to a shared origin, rather than serendipitous re-invention. The symbol is then assigned the earliest possible origin that is compatible with all the matches. We call this approach the *earliest plausible attribution* approach. We show that earliest plausible attribution yields a more natural content attribution than other approaches, including approaches based on longest matches with previous content, or approaches inspired by the edit-analysis work of Tichy [14]. By comparing new revisions with the full set of previous revisions, the earliest plausible attribution approach achieves resistance to page deletions and vandalism. As Figure 8 (A0 vs. A1) later in the paper indicates, the resulting attribution differs by over 75% from the attribution computed via comparisons to the most recent revision only, the difference being due chiefly to deletion-reinsertion attacks and other vandalism.

We introduce efficient algorithms for earliest plausible attribution. If fed all revisions at once, the algorithms can compute the content origin in time proportional to the size of the revision history, which is clearly optimal. More commonly, though, revisions are created and must be analyzed one and a time. A practical implementation must maintain a summary of all past revisions, and process a new revision on the basis of such a summary. We show that the algorithm we propose uses a summary of size proportional to all the past *change* in the previous revisions — and this change size is typically much smaller than the total revision history size, since a new revision is usually identical to the preceding one except for a few small changes. The algorithm runs in time proportional to the sum of the size of the previous summary, and the size of the most recent revision. Again, since both summary and most recent revision must be read, this is optimal.

Related work. The WikiTrust tool computes a value of reputation for Wikipedia authors and content, as well as the revision where each word was inserted [1, 2, 3]. The attribution algorithm achieves resistance to vandalism by comparing the most recent revision not only with the preceding one, but also with a set of "reference" revisions, consisting of recent revisions that either have high content reputation, or that were created by a high-reputation author. The approach is fairly effective in practice, but the attribution depends on the reputation computation: there is no independent characterization of the attribution that is computed, and the process is computationally involved. Furthermore, it is not clear how to extend the approach beyond Wikipedia.

In [3], several text matching algorithms are evaluated for their ability to explain the editing process in Wikipedia. Tichy-inspired algorithms [14] were found to be highly efficient, and as precise as any alternative, for the problem of comparing two revisions. In contrast, in this work we show that for the problem of comparing a revision with all the preceding ones, the earliest plausible attribution yields more efficient algorithms, and arguably more natural results. The problem of attributing Wikipedia content has also been studied in [6], where an algorithm based on hierarchical content matching is proposed. When a revision is compared with the preceding revision, matches of large sections of text (sections, paragraphs, sentences) are evaluated first, and a finer-grained algorithm based on the Python difflib library is used to attribute any remaining content. The resulting attribution is reported to compare favorably with the one computed by WikiTrust.

String matching is a very well studied problem; see e.g. [9] for an in-depth overview. The algorithms presented in this paper make use of several results on string matching, including tries and suffix trees. Sophisticated string matching algorithms developed for genetic applications involve a two-step process: a coarse alignment is computed between the strings, followed by a finer-grained analysis of string differences (see [9] again). This "genetic" approach is resistant to the transcription errors that occur in gene sequencing. The algorithms developed in this paper are based instead on exact matching of short sequences. It is an interesting open question whether the algorithms for attribution of revisioned content could benefit from the genetic approach.

The attribution problem considered in this paper is a special case of *information provenance* problem. For an overview of information provenance, see e.g. [4, 5] for provenance in databases, and [13, 11, 8] for an overview in a broader context.

Paper organization. After introducing some notation and concepts, we compare in Section 3 conceptual methods of defining attribution, providing justifications for our choice of earliest plausible attribution. In 4 we describe an efficient algorithm for earliest plausible attribution, and we prove that the algorithm is optimal. We present empirical results obtained in the analysis of the English Wikipedia in Section 5, and we conclude with some discussion of the results and possible future work in Section 6. All the code and data for the algorithms can be found at `https://sites.google.com/a/ucsc.edu/luca/the-wikipedia-authorship-project`.

2. DEFINITIONS

Revisions. We model revisioned content as a sequence of revisions $\overline{\rho} = \rho_0, \rho_1, \rho_2, \ldots$. Each revision ρ consists in a sequence of tokens $t_0, t_1, \ldots, t_{m-1}$, taken from a set $\mathcal{T}$ of tokens, where $len(\rho) = m$ is the length of ρ. We assume that $len(\rho_0) = 0$, so that ρ_0 represents the initial, empty revision that exists before any subsequent revision is created. For $\rho = t_0, t_1, \ldots, t_{m-1}$, we indicate with $\rho[i]$ the token t_i, and we write $\rho[i : j]$ for $t_i, t_{i+1}, \ldots, t_{j-1}$. Depending on the application, the tokens can be individual unicode characters, or they can be words in a text, tokens of a programming language, and so forth. Given a sequence $\overline{\rho}$ of revisions, a *global position* is a pair (n, k) with $n \geq 0$ and $0 \leq k < \mathit{len}(\rho_k)$.

Matches. A *match* $M = (n, i, j, n', i', j')$ between positions $[i..j-1]$ of revision ρ_n and positions $[i'..j'-1]$ of revision $\rho_{n'}$, denoted informally (and more intuitively) as $M = (\rho_n[i : j] = \rho_{n'}[i' : j'])$, consists of two revisions ρ_n, $\rho_{n'}$, and indices $0 \leq i < j \leq \mathit{len}(\rho_n)$, $0 \leq i' < j' \leq \mathit{len}(\rho_{n'})$, such that:

- $j - i = j' - i' > 0$, so that the matched portions have equal length and are non-empty;
- for all $0 \leq k < j - i$, we have $\rho_n[i+k] = \rho_{n'}[i'+k]$, so that tokens at corresponding positions of ρ_n and $\rho_{n'}$ match.

Given a match $M = (\rho_n[i : j] = \rho_{n'}[i' : j'])$, we denote by $\mathit{len}(M) = j - i$ its length. We say that a position k is *matched* by M if $i \leq k < j$. For a position k matched by M, we let $M(n, k) = (n', k - i + i')$: thus, we think of matches as partial functions between global positions that relate positions filled by equal tokens. We denote by $\mathcal{M}(\rho_n, \rho_m)$ the set of all matches between revisions ρ_n and ρ_m. We say that a match $M = (\rho_n[i : j] = \rho_{n'}[i' : j'])$ is a *sub-match* of $M' = (\rho_n[\hat{\ss} : \hat{j}] = \rho_{n'}[\hat{\ss}' : \hat{j}'])$ if $\hat{\ss} \geq i$, $\hat{j} \leq j$, and $\hat{\ss} - i = \hat{\ss}' - i'$; we say that the sub-match is *proper* if at least one of the two inequalities is strict.

Interesting matches. Our interest in matches is due to the fact that a match between a later revision and earlier one may indicate that the content of the later revision originated in the earlier one. Not all matches correspond to a common origin of the content, however. For instance, in English, the two-word sequence “such that” is very common, and it would be unreasonable to assume that they have been copied from an earlier revision whenever they appear in a later one. In order to use matches to study authorship, we need to distinguish fortuitous matches from those that indicate shared origin. An in-depth approach would likely require a probabilistic model of content structure, and of how content propagates from one revision to the next. Such a model could then be used to compute, for each revision token, a probability distribution over the places where the content might have originated.

We follow a simpler, discrete approach, where content is attributed deterministically to a revision of origin. Deterministic attribution leads to efficient algorithms that can scale to very large bodies of content, such as Wikipedia. We also remark that users of authorship information generally expect a deterministic attribution: Wikipedia visitors and editors want to know who wrote what, and copyright is based on deterministic, not probabilistic attribution. Probabilistic attribution algorithms, and the question of their efficient implementation at scale and possible accuracy advantages, remain a topic for future work.

Consider a match $M = (\rho_n[i : j] = \rho_k[i' : j'])$ between two revisions ρ_n and ρ_k, with $k < n$. To decide whether to attribute the sequence $\sigma = \rho[i], \rho[i+1], \ldots, \rho[j-1]$ to ρ_n or ρ_k, we use a *rarity* function $\gamma : \mathcal{T}^* \mapsto \mathbb{R}^+$: intuitively, the larger $\gamma(\sigma)$ is, the more likely it is that the sequence σ in ρ_n and ρ_k shares the same origin. We require that a rarity function γ satisfies the following two conditions:

- $\gamma(\emptyset) = 0$: the rarity of the empty sequence is zero.
- For all $\sigma \in \mathcal{T}^*$ and all $t \in \mathcal{T}$, we have $\gamma(\sigma) < \gamma(\sigma t)$: longer sequence are strictly rarer than shorter ones.

A simple choice is $\gamma(\sigma) = \mathit{len}(\sigma)$: the rarity of a sequence is equal to its length. More sophisticated rarity functions can be used: for instance, if we know the occurrence probability p_t of each token t, we can take $\gamma(t_0, t_1, \ldots, t_m) = \prod_{i=0}^{m} \frac{1}{p_{t_i}}$. Rarity functions based on the occurrence frequency of multi-token sequences could also be used.

Given a match $M = (\rho_n[i : j] = \rho_k[i' : j'])$, we define its interest $\gamma(M) = \gamma(\rho_n[i], \rho_n[i+1], \ldots, \rho_n[j-1])$ to be equal to the rarity of the matched sequence of tokens. We define the *interesting matches between revisions ρ_n and ρ_m, according to the rarity function γ and threshold* Δ, as the set of matches of rarity at least Δ:

$$\mathcal{M}(\rho_n, \rho_m \mid \gamma \geq \Delta) = \{M \in \mathcal{M}(\rho_n, \rho_m) \mid \gamma(M) \geq \Delta\}\,.$$

We note that if we choose $\gamma = \mathit{len}$, the set $\mathcal{M}(\rho_n, \rho_m \mid \mathit{len} \geq l)$ will consist of all matches between ρ_n and ρ_m that have length at least l. Given a position $0 \leq k < \mathit{len}(\rho_n)$ of revision ρ_n, we denote by $\mathcal{M}[k](\rho_n, \rho_m \mid \gamma \geq \Delta)$ the interesting matches between ρ_n and ρ_m that have interest at least Δ as measured by γ, and that match position k of ρ_n.

Origin labeling. An origin labeling associates with each token the revision where the token originated. Precisely, an *origin labeling* Θ for a (finite or infinite) sequence $\overline{\rho} = \rho_0, \rho_1, \rho_2, \ldots$ of revisions is a labeling that associates with each global position (n, k) of $\overline{\rho}$ its *origin* $\Theta(n, k) \in \mathbb{N}$, with $\Theta(n, k) \leq n$. If $\Theta(n, k) = n$, we say that the token $\rho_n[k]$ is new in ρ_n.

3. CONCEPTUAL ALGORITHMS

In some instances of revisioned content, such as Google Docs, full information about the edit actions by each individual user are available. In this case, the authorship can be computed by observing directly the typing, cutting, pasting, etc, performed by each editor. In many other instances, however, we can observe only the outcome of the editing process , namely, the sequence of revisions produced by the various users. This is the case for Wikipedia, and for code repositories, since the environments where users edit the code are independent from the repositories. In these cases, we must infer authorship after the fact, by comparing the result of the editing with previous content. There is no a-priori correct way to infer authorship, as we cannot reconstruct the mental process of the editors to tell whether they are copying or reinventing. One of the contributions of this paper is to introduce the notion of *earliest plausible attribution* for revisioned content, showing that it leads to plausible attribution in practice. We remark that, even when the actions of users are observable during editing, as in Google Docs, we can never be sure whether editors are retyping a passage, copying it from paper, or reinventing it anew: earliest plausible attribution can thus be a useful notion even when edit actions are observable in detail. In this section we define earliest plausible attribution and we compare it with other attribution methods. The question of efficient implementation will be the subject of the next section.

3.1 Comparison with preceding revision

Algorithm A0 computes the origin of tokens in a revision ρ_n by comparing the revision with the previous one in the sequence. Given a sequence $\overline{\rho} = \rho_0, \rho_1, \rho_2, \ldots$ of revisions, with $\rho_0 = \emptyset$ as the initial empty revision, algorithm A0 computes an origin labeling Θ for $\overline{\rho}$ proceeding inductively on the revisions. The first revision ρ_0, being empty, has a null labeling. For each subsequent revision ρ_n, $n > 0$, algorithm A0 computes all interesting matches with the preceding revision ρ_{n-1}. Every unmatched token in ρ_n is assigned an origin label of n. Each matched token is assigned the origin label of the matching position in the previous revision; if the token had multiple matches to different positions, the token is assigned the minimum of the origin labels of the corresponding positions.

Algorithm A0 Matches with previous revision.

Input: A sequence $\overline{\rho} = \rho_0, \rho_1, \rho_2, \ldots, \rho_m$ of revisions, with $\rho_0 = \emptyset$, along with a rarity function γ and a threshold Δ.
Output: An origin labeling Θ for $\overline{\rho}$.

```
 1: for revisions $n = 1, 2, 3, \ldots$ do
 2:   for all positions $0 \leq k < len(\rho_n)$ of $\rho_n$ do
 3:     Let $\widehat{\mathcal{M}} := \mathcal{M}[k](\rho_n, \rho_{n-1} \mid \gamma \geq \Delta)$.
 4:     if $\widehat{\mathcal{M}} = \emptyset$ then
 5:       $\Theta(n, k) := n$
 6:     else
 7:       $\Theta(n, k) := \min_{M \in \widehat{\mathcal{M}}} \Theta(M(n, k))$.
 8:     end if
 9:   end for
10: end for
```

One may conceive a variant algorithm, termed Algorithm A0M, where only the most interesting match(es) for each token are considered: the idea being that the longer the match, the more likely it is to correspond to origin. Algorithms A0 and A0M may yield different labelings, as illustrated in Figure 1. In the figure, we use sequence length as the rarity function, together with a threshold of 3, so that matches that are 3 or more tokens are considered interesting. In labeling symbols `b c` in ρ_3, Algorithm A0 considers two interesting matches: $(\rho_3[0 : 3] = \rho_2[0 : 43)$, involving `a b c`, and $(\rho_3[1 : 6] = \rho_2[4 : 9])$, involving `b c z z z`. The first match yields origin `1 1` for `b c`, the second `2 2`. The origin assigned by A0 is the least of these two, namely, `1 1`. Algorithm A0M, on the other hand, considers only the second match, as it is longer, and assigns to `b c` origin `2 2`.

This example highlights why we prefer to consider all interesting matches, rather than just the longest ones: even though `a b c` in ρ_3 matches `a b c` in ρ_1, it is assigned origin `1 2 2` according to A0M. We take the point of view that a match that is interesting (with a matched sequence of tokens that is sufficiently rare) denotes a common origin of the content. If there is more than one interesting match for a token position, we look at all such interesting matches as possible explanations for the origin of the content, and we err on the side of the oldest possible attribution, yielding the min in line 7 of Algorithm A0.

Algorithm A0M Origin via most interesting matches with previous revision.

Input: A sequence $\overline{\rho} = \rho_0, \rho_1, \rho_2, \ldots, \rho_m$ of revisions, with $\rho_0 = \emptyset$, along with a rarity function γ and a threshold Δ.
Output: An origin labeling Θ for $\overline{\rho}$.

```
 1: for revisions $n = 1, 2, 3, \ldots$ do
 2:   for all positions $0 \leq k < len(\rho_n)$ of $\rho_n$ do
 3:     Let $\widehat{\mathcal{M}} := \mathcal{M}[k](\rho_n, \rho_{n-1} \mid \gamma \geq \Delta)$.
 4:     if $\widehat{\mathcal{M}} := \emptyset$ then
 5:       $\Theta(n, k) = n$
 6:     else
 7:       Let $\widetilde{\mathcal{M}} = \arg\max_{M \in \widehat{\mathcal{M}}} \gamma(M)$.
 8:       $\Theta(n, k) := \min_{M \in \widetilde{\mathcal{M}}} \Theta(M(n, k))$.
 9:     end if
10:   end for
11: end for
```

ρ_3: $\mathtt{a}^1\ \mathtt{b}^1\ \mathtt{c}^1\ \mathtt{z}^2\ \mathtt{z}^2\ \mathtt{z}^2$

ρ_2: $\mathtt{a}^1\ \mathtt{b}^1\ \mathtt{c}^1\ \mathtt{x}^2\ \mathtt{b}^2\ \mathtt{c}^2\ \mathtt{z}^2\ \mathtt{z}^2\ \mathtt{z}^2$

ρ_1: $\mathtt{a}^1\ \mathtt{b}^1\ \mathtt{c}^1$

(a) A0

ρ_3: $\mathtt{a}^1\ \mathtt{b}^2\ \mathtt{c}^2\ \mathtt{z}^2\ \mathtt{z}^2\ \mathtt{z}^2$

ρ_2: $\mathtt{a}^1\ \mathtt{b}^1\ \mathtt{c}^1\ \mathtt{x}^2\ \mathtt{b}^2\ \mathtt{c}^2\ \mathtt{z}^2\ \mathtt{z}^2\ \mathtt{z}^2$

ρ_1: $\mathtt{a}^1\ \mathtt{b}^1\ \mathtt{c}^1$

(b) A0M

Figure 1: A sequence of revisions, with origin labeled according to algorithms A0 and A0M. We represent each revision by its list of tokens, using letters to denote tokens. The origin labels are computed for a rarity function equal to sequence length, and threshold of 3. We write above every token the origin that the algorithm assigns to it.

3.2 Earliest plausible attribution

Algorithm A0 (and A0M) relies on comparisons with the immediately preceding revision only. In many relevant examples of versioned content, content can be deleted from one revision only to reappear several revisions later. For instance, the content of Wikipedia pages is frequently deleted by vandals. If authorship is determined via a comparison with the immediately preceding revision only, then an editor who restores the contents of a Wikipedia page after it is deleted would be attributed the authorship of all the restored content. As these periodic acts of vandalism that destroy most of a page's content are common on Wikipedia, authorship algorithms that are based only on comparisons with the immediately preceding revision will grossly misattribute content, as we will show experimentally in Section 5.

Our preferred algorithm for attribution of revisioned content, Algorithm A1, compares the latest revision with *all the previous revisions,* looking for matches with any prior content, rather than just content in the immediately preceding revision. We call this process *earliest plausible attribution,* since the attribution it produces is the earliest that is compatible with an explanation by interesting matches. Figures 2 and 3 provides a comparison of algorithm A0 and A1 in presence of a delete-and-restore attack, as common on Wikipedia, and of a more complex attack involving content that is deleted, then gradually re-instated.

3.3 Tichy-based matching

One of the better-known algorithms for generating edit differences between revisions is due to Tichy [14]. Since the Tichy algorithm performs well in explaining the edit his-

Algorithm A1 Origin via interesting matches with all preceding revisions.

Input: A sequence $\overline{\rho} = \rho_0, \rho_1, \rho_2, \ldots, \rho_m$ of revisions, with $\rho_0 = \emptyset$, along with a rarity function γ and a threshold Δ.
Output: An origin labeling Θ for $\overline{\rho}$.

```
for revisions $n = 1, 2, 3, \ldots$ do
    for all positions $0 \leq k < len(\rho_n)$ of $\rho_n$ do
        Let $\widehat{\mathcal{M}} := \bigcup_{0<m<n} \mathcal{M}[k](\rho_n, \rho_m \mid \gamma \geq \Delta)$.
        if $\widehat{\mathcal{M}} = \emptyset$ then
            $\Theta(n, k) := n$
        else
            $\Theta(n, k) := \min_{M \in \widehat{\mathcal{M}}} \Theta(M(n, k))$.
        end if
    end for
end for
```

ρ_4: a^4 b^4 c^4 x^4 f^4 g^4 h^4

ρ_3: p^3 q^3

ρ_2: a^1 b^1 c^1 x^2 f^1 g^1 h^1

ρ_1: a^1 b^1 c^1 f^1 g^1 h^1

(a) A0

ρ_4: a^1 b^1 c^1 x^2 f^1 g^1 h^1

ρ_3: p^3 q^3

ρ_2: a^1 b^1 c^1 x^2 f^1 g^1 h^1

ρ_1: a^1 b^1 c^1 f^1 g^1 h^1

(b) A1

Figure 2: A sequence of revisions, with origin labeled according to algorithms A0 and A1, with rarity equal to length and threshold 3. This sequence illustrates a delete-and-restore event, common on Wikipedia.

tory of Wikipedia [3], it is of interest to adapt it to origin computation and compare it to A1. Given a revision $\rho_n = t_0, t_1, \ldots, t_{m-1}$, the Tichy-based Algorithm A2 searches revisions $\rho_0, \ldots, \rho_{n-1}$ for the longest prefix of $t_0, t_1, \ldots, t_{m-1}$. If this longest prefix is, say, $t_0, t_1, \ldots, t_k$, with $k \leq m - 1$ and $\gamma(t_0, t_1, \ldots, t_k) > \Delta$ for the chosen rarity function γ and threshold Δ, then the algorithm fixes the origin of $t_0, t_1, \ldots, t_k$ in ρ_n according to the origin of the matching tokens (taking the minimum, in case the longest prefix appears multiple times). The algorithm then proceeds searching for the longest prefix of the remaining unlabeled portion $t_{k+1}, t_{k+2}, \ldots, t_{m-1}$. If no longest prefix can be found, or if the longest prefix from t_0 has rarity below the threshold, then t_0 is labeled as having origin n, or $\Theta(n, 0) := n$, and the search continues from the remaining unlabeled portion $t_1, t_2, \ldots, t_{m-1}$. The process continues until the whole of ρ_n has been labeled according to its origin.

Figure 4 compares the origin labelings computed by Algorithms A1 and A2. We see that Algorithm A2 attributes to the tokens `c d a m` in ρ_4 origins `2 2 4 4`, even though these tokens constituted the first revision $\rho 1$. The attribution `1 1 1 1` computed by A1 seems more appropriate.

3.4 Properties

Given a sequence of revisions $\rho_0, \rho_1, \rho_2, \ldots$ and two origin labelings Θ, Θ', we write $\Theta \leq \Theta'$ if $\Theta(n, k) \leq \Theta'(n, k)$ at all positions n, k of the sequence; we write $\Theta < \Theta'$ if $\Theta \leq \Theta'$, and if there is at least a position (n, k) where $\Theta(n, k) < \Theta'(n, k)$. The following property establishes that, among A0, A0M, and A1, Algorithm A1 computes the earliest attribution and A0M the latest.

ρ_6: a^4 b^4 c^4 d^4 e^6 x^6 w^6 g^6 h^6 l^6

ρ_5: q^3 r^3 a^4 b^4 c^4 d^4 f^5 g^5

ρ_4: p^3 q^3 r^3 a^4 b^4 c^4 d^4

ρ_3: p^3 q^3 r^3

ρ_2: a^1 b^1 c^1 d^1 e^1 x^2 f^1 g^1 h^1 l^1

ρ_1: a^1 b^1 c^1 d^1 e^1 f^1 g^1 h^1 l^1 m^1

(a) A0

ρ_6: a^1 b^1 c^1 d^1 e^1 x^2 w^6 g^1 h^1 l^1

ρ_5: q^3 r^3 a^1 b^1 c^1 d^1 f^5 g^5

ρ_4: p^3 q^3 r^3 a^1 b^1 c^1 d^1

ρ_3: p^3 q^3 r^3

ρ_2: a^1 b^1 c^1 d^1 e^1 x^2 f^1 g^1 h^1 l^1

ρ_1: a^1 b^1 c^1 d^1 e^1 f^1 g^1 h^1 l^1 m^1

(b) A1

Figure 3: A sequence of revisions, with origin labeled according to algorithms A0 and A1, with rarity equal to length and threshold 3. In this sequence, content is first deleted and replaced with spam, then almost entirely restored.

ρ_4: a^3 b^3 c^1 d^1 a^1 m^1

ρ_3: a^3 b^3 c^2 d^2 g^2 h^2

ρ_2: c^2 d^2 g^2 h^2

ρ_1: c^1 d^1 a^1 m^1

(a) A1

ρ_4: a^3 b^3 c^2 d^2 a^4 m^4

ρ_3: a^3 b^3 c^2 d^2 g^2 h^2

ρ_2: c^2 d^2 g^2 h^2

ρ_1: c^1 d^1 a^1 m^1

(b) A2

Figure 4: A sequence of revisions, as labeled by Algorithms A1 and A2 with rarity equal to length and threshold 3.

PROPERTY 1. *Let Θ_{A0}, Θ_{A0M}, and Θ_{A1} be origin labelings computed by Algorithms A0, A0M, and A1 respectively for a sequence of revisions. Then, $\Theta_{A1} \leq \Theta_{A0} \leq \Theta_{A0M}$. Moreover, there are sequences of revisions for which each of two above inequalities is strict.*

PROOF. The weak inequalities follow from the fact that, in deriving the label of a token, the matches considered by A0M are a subset of those considered by A0, which are in turn a subset of those considered by A1. The fact that the inequalities can be strict is witnessed by Figure 1 and 2. ∎

If a portion of content with rarity above the threshold occurs twice in the revision history, Algorithm A1 will assign to the later occurrence an origin that is no later than that assigned to the earliest occurrence, and that can in fact be strictly smaller.

PROPERTY 2. *Consider a sequence of revisions $\rho_0, \ldots, \rho_i, \ldots, \rho_k$, and a match $M = (\rho_i[l : m] = \rho_k[l' : m'])$ that is sufficiently interesting. Let Θ be the origin labeling computed by A1. Then, $\Theta(k, m + j) \leq \Theta(i, m' + j)$ for all $0 \leq j < m - l$, and there are cases where the inequality can be strict.*

Algorithm A2 Origin via Tichy matching with all preceding revisions.

Input: A sequence $\overline{\rho} = \rho_0, \rho_1, \rho_2, \ldots, \rho_m$ of revisions, with $\rho_0 = \emptyset$, along with a rarity function γ and a threshold Δ.
Output: An origin labeling Θ for $\overline{\rho}$.

```
for revisions $n = 1, 2, 3, \ldots$ do
  $k := 0$
  while $k < len(\rho_n)$ do
    Search in $\rho_0, \ldots, \rho_n$ for the longest matching prefixes of $t_k, t_{k+1}, \ldots, t_{len(\rho_n)-1}$. Let $t_k, \ldots, t_m$ be the longest matched prefix, and let $\mathcal{A} = \{(n_1, k_1), \ldots, (n_p, k_p)\}$ be the (possibly empty) set of pairs where the longest matches occur.
    if $\mathcal{A} \neq \emptyset \wedge \gamma(t_k, \ldots, t_m) \geq \Delta$ then
      for $i \in \{0, 1, \ldots, m-k\}$ do
        $\Theta(n, k+i) := \min_{1 \leq j \leq p} \Theta(n_j, k_j + i)$
      end for
      $k := m + 1$
    else
      $\Theta(n, k) := n$
      $k := k + 1$
    end if
  end while
end for
```

ρ_5: $\mathtt{a}^3$ $\mathtt{b}^3$ $\mathtt{c}^1$ $\mathtt{d}^1$ $\mathtt{g}^2$ $\mathtt{h}^2$

ρ_4: $\mathtt{a}^3$ $\mathtt{b}^3$ $\mathtt{c}^1$ $\mathtt{d}^1$ $\mathtt{a}^1$ $\mathtt{m}^1$

ρ_3: $\mathtt{a}^3$ $\mathtt{b}^3$ $\mathtt{c}^2$ $\mathtt{d}^2$ $\mathtt{g}^2$ $\mathtt{h}^2$

ρ_2: $\mathtt{c}^2$ $\mathtt{d}^2$ $\mathtt{g}^2$ $\mathtt{h}^2$

ρ_1: $\mathtt{c}^1$ $\mathtt{d}^1$ $\mathtt{a}^1$ $\mathtt{m}^1$

Figure 5: A sequence of revisions, as labeled by Algorithm A1 with rarity equal to length and threshold 3. Note that $\rho_5 = \rho_3$, yet the origin labels for some tokens in ρ_5 are smaller than the corresponding ones in ρ_3.

PROOF. The result follows from the fact that the matches for $\rho_k[l' : m']$ include $\rho_i[l : m]$. The fact that the inequality can be strict is illustrated in Figure 5. ∎

This property formalizes the robustness to attacks of Algorithm A1. Consider a good revision ρ_i of a page. If vandals produce revisions $\rho_{i+1}, \rho_{i+2}, \ldots, \rho_{k-1}$, and the page is then restored to its good state $\rho_k = \rho_i$, the property ensures that no content in $\rho_i = \rho_k$ becomes attributed to an author of $\rho_{i+1}, \rho_{i+2}, \ldots, \rho_{k-1}, \rho_k$.[2]

4. EFFICIENT ALGORITHMS

In the previous section, we presented various conceptual algorithms for attributing origin to versioned content. The algorithms presented there are extremely inefficient, and have conceptual value only. In this section, we examine the question of efficient implementation for these conceptual algorithms.

Input size and change size. Given a sequence of revisions $\overline{\rho} = \rho_0, \rho_1, \ldots, \rho_n$, the input size for our attribution algoirthms is $|\overline{\rho}| = \sum_{i=0}^{n} len(\rho_i)$ (assuming that tokens can be represented in constant space). In revisioned content, it is often the case that only a small portion of the content is modified at each revision, so that consecutive revisions differ only in a few tokens. It is thus insightful to study the performance of the algorithms not only as a function of the size of the input, but also as a function of the size of the change that occurred. To this end, given two consecutive revisions ρ, ρ', we define $\Delta(\rho, \rho') = \sum_{i=1}^{m} |\beta_i| + \sum_{i=1}^{m} |\gamma_i|$, where $\beta_1, \ldots, \beta_k$ and $\gamma_1, \ldots, \gamma_m$ are the shortest sequences so that we can write:

$$\rho = \alpha_0 \beta_1 \alpha_1 \beta_2 \alpha_2 \cdots \beta_n \alpha_n$$
$$\rho' = \alpha_0 \gamma_1 \alpha_1 \gamma_2 \alpha_2 \cdots \gamma_n \alpha_n$$

In other words, we write ρ and ρ' as composed of maximal sequences of unchanged portions of text $\alpha_0, \ldots, \alpha_m$, and of portions $\beta_1, \ldots, \beta_m$ in ρ that will be replaced by sequences $\gamma_1, \ldots, \gamma_m$ in ρ'. We then define the *change size change*$(\overline{\rho})$ of $\rho_0, \rho_1, \ldots, \rho_n$ as *change*$(\overline{\rho}) = \sum_{i=0}^{n-1} \Delta(\rho_i, \rho_{i+1})$.

Summary size and one-revision update. Revisioned content is produced, as the name implies, one revision at a time. When computing the origin of the tokens in the newest revision ρ_n, it would be impractical to read and re-process all previous revisions $\rho_0, \ldots, \rho_{n-1}$. Practical algorithms rely on a *summary* $\mathcal{S}_{n-1}$ of $\rho_0, \ldots, \rho_{n-1}$, containing all the information that the algorithm needs to know about the preceding revisions to attribute later revisions. The algorithms compute the origin labeling for ρ_n on the basis of $\mathcal{S}_{n-1}$ and ρ_n, producing as output both $\mathcal{S}_n$ and the origin labeling for ρ_n. We refer to this computation as the *one-revision update.* We will thus study how the summary size, and the running time for the one-revision update depend on the input size and change size.

4.1 Algorithm A3

Consider a fixed a rarity function γ and a rarity threshold Δ. We say that a sequence of tokens $t_1, t_2, \ldots, t_n$ is *minimally interesting* if $\gamma(t_1, t_2, \ldots, t_n) \geq \Delta$, and at least one of $\gamma(t_2, \ldots, t_n) < \Delta$ or $\gamma(t_1, \ldots, t_{n-1}) < \Delta$ holds. When the rarity function is simply the number of tokens, and the rarity threshold Δ is an integer, then the minimally interesting sequences are the sequences consisting of Δ tokens. We say that a match $M = (\rho_n[i : j] = \rho_{n'}[i' : j'])$ is *minimally interesting* if $\rho_n[i], \rho_n[i+1], \ldots, \rho_n[j-1]$ is minimally interesting. To obtain an efficient implementation of Algorithm A1, we start from the observation that in Step 3 of Algorithm A1, we need to consider only minimally interesting matches.

LEMMA 1. *If in Step 3 of Algorithm A1 the set $\widehat{\mathcal{M}}$ is limited only to minimally interesting matches, the labeling computed by the algorithm is unchanged.*

PROOF. For a token t_k of ρ_n, let M be a match realizing the minimum in Step 7 of Algorithm A1, and let $t_i, \ldots, t_j$ be the matched sequence, with $i \leq k \leq j$. If M is minimally interesting, the result holds. If M is not minimally interesting, then both sub-matches for $t_i, \ldots, t_{j-1}$ and $t_{i+1}, \ldots, t_j$ are interesting, and t_k belongs to one of them. Continuing in this fashion, we can find a submatch M' of M that contains t_k and that is minimally interesting. Since t_k would be assigned the same origin under M or M', the result holds. ∎

[2] This is true provided that the content of ρ_k forms an interesting match with the content of ρ_i. This can be ensured by adding start and end markers to the text of revisions, and by considering interesting any match that contains such markers.

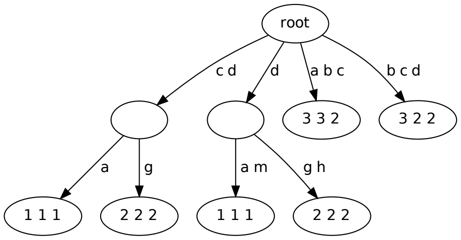

Figure 6: Trie resulting after processing revisions ρ_1, ρ_2, ρ_3 as in Figure 4.

This result suggests implementing Algorithm A1 in terms of a *trie.* A trie $\mathcal{T}$ is a tree whose edges are labeled with tokens, and such that the edges outgoing from a node are labeled by distinct tokens. We say that a sequence of tokens $t_1, t_2, \ldots, t_m$ *belongs* to the trie $\mathcal{T}$, written $t_1, t_2, \ldots, t_m \in \mathcal{T}$, if there is a path from the root labeled with the sequence, and we use the sequence to refer to the node where the path ends. If the sequence $t_1, t_2, \ldots, t_m$ is minimally interesting, we say that the corresponding node is minimally interesting. If $\gamma = \mathit{len}$ and Δ is an integer, the minimally interesting nodes are those at depth Δ in the trie. We denote by $\perp$ the empty trie consisting only of a root node, and we denote by $\mathit{Ins}(\mathcal{T}; t_1, \ldots, t_m)$ the result of creating a path labeled by $t_1, \ldots, t_m$ in $\mathcal{T}$ in the trie. In the implementation of A1, we use tries to represent all the minimally interesting sequences of tokens that have occurred in past revisions. Each minimally interesting node $t_1, \ldots, t_m$ of the trie is labeled with the origin $k_1, \ldots, k_m = \ell(t_1, \ldots, t_m)$ of the sequence of tokens $t_1, \ldots, t_m$. This yields Algorithm A3.

Figure 6 illustrates the trie resulting after processing revisions ρ_1, ρ_2, ρ_3 as in Figure 4, for a rarity function equal to length, and threshold 3. The leaf nodes are the minimally interesting nodes. To save space in the trie, we omit the non-interesting nodes that have a single child, concatenating the labels of the edges leading into and out of such nodes.

The following theorem shows that Algorithms A3 and A1 compute the same origin labels.

THEOREM 1. *Algorithm A3 computes the same origin labels as Algorithm A1.*

To state the proof of this theorem, consider a sequence $\sigma = t_1, \ldots, t_k$ occurring at least once in a set of revisions $\rho_0, \ldots, \rho_m$ that has been labeled according to its origin by Algorithm A1. For $1 \leq j \leq k$, let p_j be the minimum label that token t_j is assigned in any of these occurrences. We say that $p_1, \ldots, p_k$ is the *minimal labeling* of σ in $\rho_0, \ldots, \rho_m$.

PROOF. The proof proceeds by induction, using the inductive hypothesys that, after processing revisons $\rho_0, \ldots, \rho_n$, the trie $\mathcal{T}$ contains exactly all the minimally interesting sequences occurring in $\rho_0, \ldots, \rho_n$, each labeled with its minimal labeling in $\rho_0, \ldots, \rho_n$.

Assume that Algorithm A3 has processed $\rho_0, \ldots, \rho_{n-1}$ already, and is processing ρ_n.

First, we show that this inductive hypothesis implies that algorithms A1 and A3 produce the same labeling. There are two directions to the argument.

- Assume that Algorithm A1 assigns origin label p to token $\rho_n[k]$. Let $M = (\rho_n[i : j] = \rho_{n'}[i' : j'])$ be the minimally interesting match for which the minimum in Line 7 is realized (this exists, due to Lemma 1). By induction hypothesis, the trie $\mathcal{T}$ will contain the sequence $\rho_{n'}[i' : j']$ with its minimal labeling, in which the token $\rho_n[k]$ is labeled with origin p. Thus, Algorithm A3 in Steps 3–13 will assign to $\rho_n[k]$ an origin no larger than p.
- Conversely, assume that Algorithm A3 assigns origin p to token $\rho_n[k]$. Then, $\mathcal{T}$ must have contained a minimally interesting sequence $\rho_n[j : l] = t_j, \ldots, t_{l-1}$, for $j \leq k < l$, where t_k is labeled by p. By inducton hypothesis, p is the minimal label of t_k in all occurrences of $t_j, \ldots, t_{l-1}$ in $\rho_0, \ldots, \rho_{n-1}$, indicating that Algorithm A1 also labels $\rho_n[k]$ with label no greater than p.

Algorithm A3 Implementation of A1 in terms of tries.

Input: A sequence $\overline{\rho} = \rho_0, \rho_1, \rho_2, \ldots, \rho_m$ of revisions, with $\rho_0 = \emptyset$, along with a rarity function γ and a threshold Δ.
Output: An origin labeling Θ for $\overline{\rho}$.

```
 1: $\mathcal{T} := \perp$
 2: for revisions $n = 1, 2, 3, \ldots$ do
 3:   for all positions $0 \leq k < \mathit{len}(\rho_n)$ of $\rho_n$ do
 4:     $\Theta(n, k) := n$
 5:   end for
 6:   for all minimally interesting sequences $t_k, \ldots, t_m$ of $\rho_n$ do
 7:     if $t_k, \ldots, t_m \in \mathcal{T}$ then
 8:       $i_k, \ldots, i_m := \ell(t_k, \ldots, t_m)$
 9:       for all $j \in [k, \ldots, m]$ do
10:         $\Theta(n, j) := \min\{\Theta(n, j), i_j\}$
11:       end for
12:     end if
13:   end for
14:   for all minimally interesting sequences $t_k, \ldots, t_m$ of $\rho_n$ do
15:     if $t_k, \ldots, t_m \in \mathcal{T}$ then
16:       $\ell(t_k, \ldots, t_m) := \Theta(n, k), \ldots, \Theta(n, m)$
17:     else
18:       $\mathcal{T} := \mathit{Ins}(\mathcal{T}; t_k, \ldots, t_m)$
19:       $\ell(t_k, \ldots, t_m) := \Theta(n, k), \ldots, \Theta(n, m)$
20:     end if
21:   end for
22: end for
```

Second, we show that once ρ_n is processed by A3, the induction hypothesis holds also for $\rho_0, \ldots, \rho_n$. Consider a minimally interesting sequence σ occurring in ρ_n (the situation of minimally interesting sequences not occurring in ρ_n is unchanged). The arguments in the first part of this proof ensure that once Steps 3–13 have terminated, the sequence σ in ρ_n is labeled according to its minimal labeling. Steps 14–21 ensure then that the sequence σ is present in the trie $\mathcal{T}$, and is labeled in it according to its minimal labeling. This completes the induction step. ∎

The following theorem characterizes the time and space requirements for Algorithm A3.

THEOREM 2. *If there is an integer M such that all token sequences of length at least M are interesting, then given a* *sequence $\rho_0, \rho_1, \rho_2, \ldots$ of revisions, Algorithm A3 can perform a one-revision update for revision ρ_n using a summary of size $\mathcal{O}(change(\rho_0, \ldots, \rho_{n-1}))$, and in time $\mathcal{O}(len(\rho_n))$.*

Proof. Algorithm A3 uses as summary for $\rho_0, \ldots, \rho_{n-1}$ the trie $\mathcal{T}_{n-1}$ resulting from the processing of these revisions. To prove the space requirement, we can prove by induction over n that $|\mathcal{T}_n| \leq K \cdot change(\rho_0, \ldots, \rho_n)$, for some fixed $K \geq 0$. Note that M is a bound for the length of any minimally interesting sequence: in fact, any interesting sequence σ longer than M has its leftmost M tokens, and rightmost M tokens, also form interesting sequences, contradicting the minimality of σ. Let $K = M(M+1)/2$ be the maximum number of sequences of length at most M that contain a given position. Note that a single insertion or deletion going from ρ_{n-1} to ρ_n affects at most K minimally interesting sequences in ρ_n. Therefore, at most $K \cdot \Delta(\rho_{n-1}, \rho_n)$ new minimally interesting sequences are going to be inserted in $\mathcal{T}_{n-1}$ in order to obtain $\mathcal{T}_n$. This leads to the space bound for the summary.

To prove the time bound for the processing of ρ_n, it suffices to note that there are at most $len(\rho_n)$ minimally interesting matches involving ρ_n, and that processing each one of them (including accessing the trie for retrieving the minimal labeling of any match) takes constant time (the trie has depth at most K). ∎

Note that the theorem implies that the processing of a sequence $\rho_0, \ldots, \rho_n$ of revisions can be done in time $\mathcal{O}(|(|\rho_0, \ldots, \rho_n))$.

If the rarity of a sequence of tokens is taken to be its length, then trivially all sequences longer than the rarity threshold are rare. Another case when the length of minimally interesting sequences of tokens is bounded is when the rarity of a sequence of tokens $t_1, \ldots, t_k$ is computed as $\gamma(t_1, \ldots, t_k) = \prod_{i=1}^{k} \frac{1}{p_{k_i}}$ for some token probabilities $0 \leq p_{k_i} \leq 1$, and if there is an upper bound $c < 1$ for the probability of any token.

These results suggest that Algorithm A3 is optimal: it is not possible to label a sequence of revisions in time less than the input size, and it is not possible to label a new revision storing less information about the past than all change that has occurred (except if compression techniques are used; such techniques can also be applied to the representation of our trie summaries).

In large-scale implementations of origin analysis, the summary of a revision sequence cannot be stored permanently in-memory: rather, it must be read from persistent storage (such as a database) before the algorithm analyzes a new revision, and written back to persistent storage once the analysis is done. If the time to read and write the summary is included, then the time required for analyzing revision ρ_n of sequence $\rho_0, \rho_1, \rho_2, \ldots$ is in $\mathcal{O}(len(\rho_n) + change(\rho_0, \ldots, \rho_n))$.

4.2 Tichy matching

The Tichy-based Algorithm A2 is defined in terms of longest common matches. We can obtain an efficient implementation in terms of *suffix trees,* which provide the most time efficient implementation of the longest common substring problem [9]. A suffix tree is a tree-like data structure that can represent all the suffixes $a_i, a_{i+1}, a_{i+2}, \ldots, a_m$, $0 \leq i < m$, of a given string $a_0, a_1, \ldots, a_m$; they can be constructed in time linear in the length of the string [16, 10, 15]. The construction of suffix trees can be adapted so that $\mathcal{S}_{n-1}$ is a suffix tree representing all the suffixes of $\rho_0, \rho_1, \ldots, \rho_{n-1}$; see [9] for similar adaptations. The origin information can be associated with the suffix tree in similar fashion to what was done for the trie; we omit the details to conserve space. The drawback of this algorithm, compared to A3, is that the size required by the summary is proportional to the size of all previous revisions, rather than to the change size. This because a change involving a token in the middle of a revision of length m gives rise to $m/2$ new suffixes on average, each of which corresponds to at least one new suffix tree node.

Theorem 3. *Let $M = |\rho_0, \ldots, \rho_n|$ and $D = change(\rho_0, \ldots, \rho_n)$. The suffix-tree-based implementation of Algorithm A2 produces the origin labels for revision ρ_n in time $\mathcal{O}(M)$; the time for labeling the complete sequence $\rho_0, \ldots, \rho_n$ is $\mathcal{O}(M^2)$. There are some examples of input for which the running time for ρ_n exceeds $K \cdot D$, for any $K \geq 0$, so that the running time is not $\mathcal{O}(D)$. The size of $\mathcal{S}_n$ is $\mathcal{O}(M)$, and is not in $\mathcal{O}(D)$.*

Proof. The space and time results are a consequence of the results on the construction of suffix trees [16, 10, 15, 9]. The existence of sequences in which the summary size is proportional to the entire input size, rather than to the change size, follows from the fact that changing a single token in a revision of length m leads to the creation of a number of new suffixes that is proportional to m (on average, equal to $m/2$). These new suffixes must be represented in the suffix tree, so that $\mathcal{S}_n$ is in $\mathcal{O}(|\rho_0, \ldots, \rho_n|)$ but not in $\mathcal{O}(change(\rho_0, \ldots, \rho_n))$. ∎

5. EXPERIMENTAL RESULTS

We have produced a robust, scalable implementation of Algorithm A3 that can be applied to very large wikis, including the English Wikipedia. Each revision is parsed in a sequence of tokens, which consists of white-space separated sequences of non-whitespace characters: tokens thus loosely correspond to words. This tokenization step could be improved by considering the structure of the MediaWiki markup language. We do not use individual (unicode) characters as our unit of tokenization, for two reasons. First, using words as attribution units tends to produce more natural results, since contributors typically create or rearrange content in word units; word-level attribution is also easier to display via coloring or other visual cues. Second, using individual characters as tokens would lead to a larger size for the trie summary, as the trie would grow deeper.

The algorithm uses as rarity function the length of a token sequence, and a configurable threshold. The algorithm does not use a rarity function that depends on token (word) frequency, chiefly to save space by avoiding the need to store the frequency of a large number of words; we may revisit this decision at a later time. For each wiki page $\mathcal{P}$, the algorithm stores in persistent storage the pair $(n, \mathcal{T}_n)$, consisting of the index n of the last revision of $\mathcal{P}$ that has been processed, along with the labeled trie $\mathcal{T}_n$ representing the summary. When a new revision ρ_m for $\mathcal{P}$ is produced, with $m > n$, the algorithm processes all revisions $\rho_{n+1}, \ldots, \rho_m$: there can be multiple revision to analyze, since the algorithm may have been inactive at times (due to system maintenance), or indeed, it may not have run yet on the page. Each of $\rho_{n+1}, \ldots, \rho_m$ is fed to the algorithm; the algorithm

	attribution difference	trie size
$\Delta_N = 200$, $\Delta_T = 180$ days	1.3 %	70 %
$\Delta_N = 100$, $\Delta_T = 90$ days	2.1 %	55 %

Figure 7: Attribution difference, and trie size, for various aging thresholds, as compared to no content aging.

computes and stores the origin of these revisions, and finally stores $(m, \mathcal{T}_m)$ associated with page $\mathcal{P}$.

The code, and a demo of this implementation is available at `https://sites.google.com/a/ucsc.edu/luca/the-wikipedia-authorship-project`, along with all the data used for the experiments reported here. We provide experimental data computed on two revison datasets:

- Dataset A: articles with more than 200 revisions in files wiki-00000066.xml.gz and wiki-00000193.xml.gz. The dataset consists of 78k revisions in 75 articles.
- Dataset B: articles with at least 1000 revisions occurring in files wiki-00000066.xml.gz, wiki-00000193.xml.gz, wiki-00000134.xml.gz and wiki-00000384.xml.gz. The dataset consists in 50 revisions.

The above *.xml.gz files were chosen at random among the first 1000 files obtained by splitting in 100-page portions a 2010 dump of the English Wikipedia. Unless otherwise noted, we provide results for a rarity function equal to length, and threshold 4.

Content aging. In the editing of Wikipedia revisions, it occasionally happens that vandals introduce vast amounts of spurious content. This content is almost immediately removed by editors or non-vandal users. Yet, since our algorithms store a representation of the entire history of each page, that spurious content would persist indefinitely in our trie summary. This would offer an avenue to vandals for severely impacting our performance. To limit this effects of vandalism, our implementation discards content that has not appeared in any recent revision: this is acceptable in practice, since content that has been long removed from a page is unlikely to be re-inserted. To this end, we label every node of the trie $\mathcal{T}$ used in Algorithm A3 with the *node age,* consisting of a pair (N, T). The integer N and the timestamp T represents, respectively, the most recent revision index and the most recent time when the node was traversed by the algorithm. Once Algorithm A3 has processed a revision n produced at time T_n, and before writing back the trie to persistent storage, we prune from the trie all the nodes that have both $n - N > \Delta_N$ and $T_n - T > \Delta_T$, where the thresholds Δ_N and Δ_T are configurable. Table 7 compares the difference in attribution and size arising from different aging thresholds. The table was obtained from dataset A.

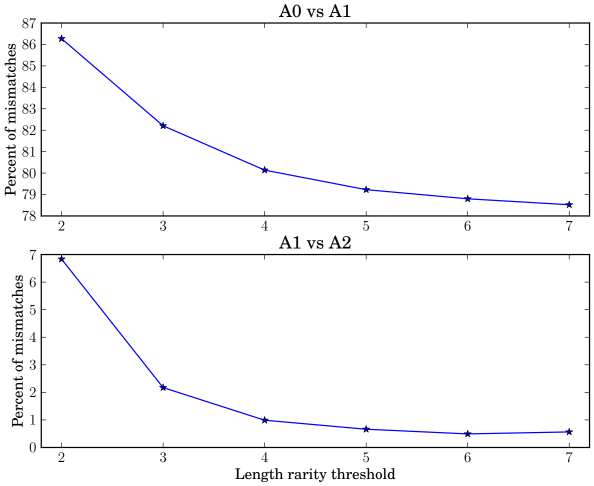

Figure 8: Difference between attribution by A0, A1 and A2 for length rarity function with various threshold.

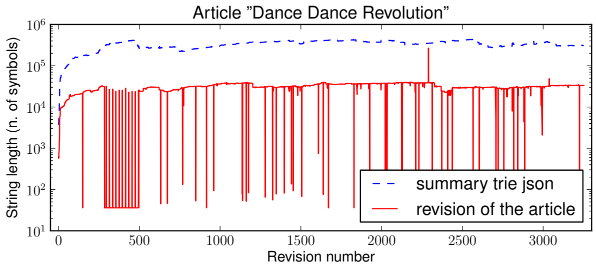

Figure 9: Length of revisions (in number of characters) of the article "Dance Dance Revolution" compare to length of json string with the trie summary. Dips in the revision size indicate content deletions due to vandalism.

Attribution comparison among A0, A1, A2. Figure 8 plots the difference in the attributions computed by Algorithms A0, A1, and A2, for a rarity function equal to token sequence length, and various values of rarity threshold. These comparisons have been done without using any age-driven pruning of trie nodes in Algorithm A1, to make the comparison fair across algorithms. The figure gives the tokens with different attribution, as percentage of the total tokens, for dataset A. As we can see, Algorithm A1 computes an attribution that is over 75% different from the one computed by Algorithm A0. This is due to the fact that Algorithm A0 considers new any content that is re-inserted after a deletion. As an example, Figure 9 plots the size of the revisions, and summary trie, for the Wikipedia article on "Dance Dance revolution"; the frequent dips in revision size correspond to content deletions by vandals. From Figure 8 we see also that the attribution difference between algorithms A1 and A2 is of only a few percentage points, when the length of minimally interesting matches is 3 or more. The advantage of Algorithm A1 over A2 lies in its efficient implementation.

Size of trie and suffix tree summaries. Figure 10 plots the ratio between the size of the trie serialized in Json, and the average size of the last 10 revisions, for aging values $\Delta_N = 100$ and $\Delta_T = 90$ days and dataset B. We use the average size of the last 10 revisions, rather than the size of the last revision, to avoid very large spikes in the ratio when the content of a revision is deleted by vandals. The average ratio is approximately 10; the ratio can be reduced to about 3 by compressing the trie serializations with gzip. This is a very practical amount of storage, which is dwarfed in the English Wikipedia by the amount of storage required to store all revisions of every page.

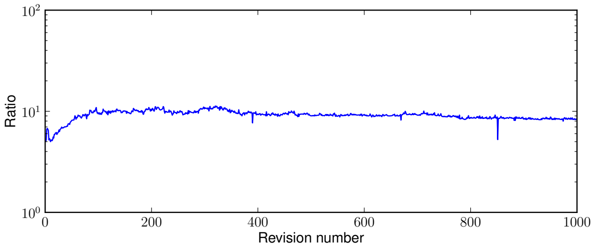

Figure 10: Ratio between the trie summary size and the average size of the last 10 revisions.

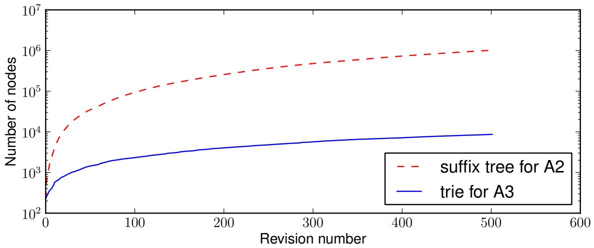

Figure 11: Size comparison between trie summaries for A3 and suffix tree summaries for A2.

In Figure 11 we compare the size of the trie summaries used by Algorithm A3, with the size of the suffix tree summaries used in implementing Algorithm A2. Dataset B was used, and no content aging was applied, to make the comparison fair. The trie sizes are tied to the change between revisions, and since we discard text that has been dead for long, they tend to be a constant multiple of the revision size. On the other hand, the suffix tree sizes are proportional to the total size of past revisions.

Time performance. Figure 12 summarizes the time performance of Algorithm A3, as a function of revision size. In the figure, the *algorithm time* is the time required by steps 3–21, as well as content aging, of A3; the *serialization time* is the time required for serializing and deserializing the trie into json. As we see, these two times are of the same order of magnitude, indicating that there is limited scope for improvement by optimizing the implementation of steps 3–21. The figure was obtained using dataset A.

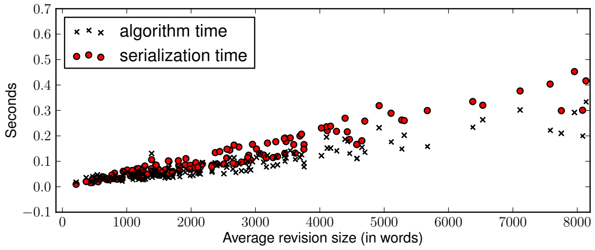

Figure 12: Time performance of algorithm A3. Each point in the plot represents an article, with the average revision size on the X axis. The times required by attribution computation, and trie serialization and deserialization, are reported on the Y axis.

6. DISCUSSION

We have considered so far revisioned content that consists in a single revisioned entity. Most revisioned content, however, consists of multiple entities: a national Wikipedia consists of a set of pages, each of which is versioned, and a code repository similarly consists of multiple files, each revisioned. Furthermore, in modern revisioning systems such as git (`http://git-scm.com`), the various revisions are organized in branches. Since code is commonly copied across files, and to a lesser extent, content is moved across Wikipedia pages, an origin analysis that spans a whole repository is often desirable.

We can perform such repository-wide analysis with the algorithms we discussed in this paper, by considering the stream of *all* revisions $\rho_0, \rho_1, \rho_2, \ldots$ in the order they are created, regardless of the entity (page, or file, or branch) to which they belong. The content of each new revision will be compared with all previous content, assigning origin via matching with corresponding occurrences. The algorithms could be improved by considering as more likely the matches that relate different versions of the same entity, as compared to matches that relate different entities. For software repositories, which are of moderate size, and where revisions are typically generated at low speed (even large industrial code bases have intervals between revisions of several seconds), such a global origin analysis would be feasible. In the English Wikipedia, however, several revisions per second may be created. From our experimental data, the size of a global summary would about ten times the cumulative size of the most recent revisions of all pages, leading to a size of approximately one terabyte. This size exceeds the RAM memory easily available in a single, low-cost host. The design of a system capable of comparing, in real time, every revision of Wikipedia with the whole of its past history would be challenging, and the result expensive to operate. For this reason, in our implementation we have opted to compare new revisions only with the previous content of the page to which the revisions belong. If required, we will address content moved across pages via specialized tools.

Compared to the algorithm of [6], the one presented here offers a simple mathematical definition of authorship, is applicable to any revisioned content, comes with compexity bounds and robustness characterization, and is well-suited to an implementation in which the authorship information needs to be computed on-line, as revisions are made. Unfortunately, we became aware of the work of [6] too late to be able to present here a quantitative comparison of how well the two algorithms match a human perception of authorship on the Wikipedia.

Acknowledgements

This work was supported in part by the NIH award 1R01GM089820-01A1, and by a gift from Google, Inc. The content is solely the responsibility of the authors and does not necessarily represent the official views of the NIH.

7. REFERENCES

[1] B. Adler and L. de Alfaro. A content-driven reputation system for the Wikipedia. In *WWW 2007, Proc. of the 16th Intl. World Wide Web Conference.* ACM Press, 2007.

[2] B. Adler, L. de Alfaro, I. Pye, and V. Raman. Measuring author contributions to the Wikipedia. In *WikiSym: International Symposium on Wikis*, 2008.

[3] B. T. Adler. *WikiTrust: Content-Driven Reputation for the Wikipedia*. PhD thesis, UC Santa Cruz, 2012.

[4] P. Buneman, S. Khanna, and T. Wang-Chiew. Data provenance: Some basic issues. In *FST TCS 2000*, Lect. Notes in Comp. Sci., pages 87–93. Springer-Verlag, 2000.

[5] P. Buneman, S. Khanna, and T. Wang-Chiew. Why and where: A characterization of data provenance. In *ICDT 2001: Intl. Conf. on Database Theory*, volume 1973 of *Lect. Notes in Comp. Sci.*, pages 316–330. Springer-Verlag, 2001.

[6] F. Flöck and A. Rodchenko. Whose article is it anyway? — Detecting authorship distribution in wikipedia articles over time with WIKIGINI. In *Proceedings of the Wikipedia Academy*. Online Publication, 2012.

[7] A. Forte and A. Bruckman. Why do people write for the Wikipedia? Incentives to contribute to open-content publishing. In *SIGGROUP 2005 Workshop: Sustaining Community*, 2005.

[8] J. Freire, D. Koop, E. Santos, and C. Silva. Provenance for computational tasks: A survey. *Computing in Science and Engineering*, 10(3), 2008.

[9] D. Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, 1997.

[10] E. McCreight. A space-economical suffix tree construction algorithm. *J. ACM*, 23:262–272, 1976.

[11] L. Moreau, P. Groth, S. Miles, J. Vazquez-Salceda, J. Ibbotson, S. Jiang, S. Munroe, O. Rana, A. Schreiber, V. Tan, and L. Varga. The provenance of electronic data. *Communications of the ACM*, 51(4), 2008.

[12] O. Nov. What motivates wikipedians? *Comm. ACM*, 50(11):60–64, 2007.

[13] Y. Simmhan, B. Plale, and D. Gannon. A survey of data provenance in e-Science. *ACM SIGMOD Record*, 34(3), 2005.

[14] W. Tichy. The string-to-string correction problem with block move. *ACM Trans. on Computer Systems*, 2(4), 1984.

[15] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14:249–260, 1995.

[16] P. Weiner. Linear pattern matching algorithms. In *Proc. of the 14th IEEE Symp. on Switching and Automata Theory*, pages 1–11, 1973.