WikiWho: Precise and Efficient Attribution of Authorship of Revisioned Content

Fabian Flöck
Institute AIFB
Karlsruhe Institute of Technology, Germany
fabian.floeck@kit.edu

Maribel Acosta
Institute AIFB
Karlsruhe Institute of Technology, Germany
maribel.acosta@kit.edu

ABSTRACT

Revisioned text content is present in numerous collaboration platforms on the Web, most notably Wikis. To track authorship of text tokens in such systems has many potential applications; the identification of main authors for licensing reasons or tracking of collaborative writing patterns over time, to name some. In this context, two main challenges arise: First, it is critical for such an authorship tracking system to be precise in its attributions, to be reliable as a data basis for further processing. Second, it has to run efficiently even on very large datasets, such as Wikipedia. As a solution, we propose a graph-based model to represent revisioned content and an algorithm over this model that tackles both issues effectively. We describe the optimal implementation and design choices when tuning it to a Wiki environment. We further present a gold standard of 240 tokens from English Wikipedia articles annotated with their origin. This gold standard was created manually and confirmed by multiple independent users of a crowdsourcing platform. It is the first gold standard of this kind and quality and our solution achieves an average of 95% precision on this data set. We also perform a first-ever precision evaluation of the state-of-the-art algorithm for the task, exceeding it by over 10% on average. Our approach outperforms the execution time of the state-of-the-art by one order of magnitude, as we demonstrate on a sample of over 240 English Wikipedia articles. We argue that the increased size of an optional materialization of our results by about 10% compared to the baseline is a favorable trade-off, given the large advantage in runtime performance.

1. INTRODUCTION

Collaborative authoring of text-based content, such as Wiki pages, shared editable documents or software code is a common sight on the web today. Most of these systems keep track of the different versions of the content created with every edit. In this paper, we propose an approach how to efficiently and precisely assign the revision of origin – and with it, its author – to particular text tokens. While line-level tracking of recent changes is a feature of many code revisioning systems (cf. Section 2), this level of attribution can prove insufficient in the case of a community that produces large amounts of collaboratively written natural language text. A word-level tracking that is proven to be trustable in its attributions and that can track reintroductions and moves of chunks of text in an acceptable runtime for end-users seems to be very useful, especially on a platform like Wikipedia, as has been previously discussed [6, 7]. While research shows that Wikipedians are motivated by the recognition by their peers that comes with authoring content [8] more practical needs also exist [6]. To reuse a Wikipedia article under the CC-BY-SA license,[1] for example, might require to list the main authors of the article, which are not easily retrievable.[2] Authorship tracking in articles can further raise awareness by editors and readers for editing dynamics, concentration of authorship [7], tracing back certain viewpoints or generally understanding the evolution of an article. Recently, Wikimedia Deutschland e.V. introduced the “Article Monitor”,[3] a gadget that aims to assist users with these exact issues and which makes use of the results of a basic authorship algorithm [7] whose general concept we use as a foundation in this work. The Wikipedia community has come up with some intended solutions related to the attribution problem, which highlights the utility of such a solution for Wikipedians.[4]

As outlined in previous work [6], the attribution problem at this fine-grained level and in highly dynamic environments like Wikipedia is not trivial, as we will discuss when introducing our content model in Section 3.1. Frequent reintroductions, content moves and other actions can be hard to monitor. In code revisioning, similar issues can emerge and finer-grained attribution techniques can have similar merits, as small changes of a few characters might have great effects, just as (re)introducing larger code chunks.

Against this background, the main contributions of this paper are the model for revisioned content we devise (Section 3), the algorithm we build upon that model (Section 4), the generation of a gold standard for precision testing (Section 5.1.1) and the experimental evaluation of precision, runtime and materialization size in comparison to the state-of-the-art [6] (remainder of Section 5).

Although we use the example of the English Wikipedia as inspiration and testing ground, the proposed model and algorithm can be understood as components of a more generally applicable method for revisioned content. We are convinced that many of the assumptions made for the use case of Wikipedia also hold true not only for other Wikis but also for other revisioned content systems.

[1] http://creativecommons.org/licenses/by-sa/2.5/
[2] https://en.wikipedia.org/wiki/Wikipedia:Reusing_Wikipedia_content
[3] http://tools.wmflabs.org/render/stools/
[4] https://en.wikipedia.org/wiki/Wikipedia:Tools#Page_histories, cf. also Keppmann et al. [10]

2. RELATED WORK

In the context of Software Engineering, content attribution has been studied in terms of code ownership. In a programming environment, lines of code is still used as a metric for measuring technical quality of source code [4], as well as a basic unit to identify contributors among software components. Therefore, solutions to identify code ownership are designed to operate on a coarse-grained level, e.g., line-by-line [13, 14]. Decentralized Source Code Management (DSCM) systems such as Apache Subversion [13] or Git[5] provide a feature to keep track of changes line-by-line. This functionality is denominated `blame` or `praise` depending on the system. Everytime a contributor performs a change on a line of code, the system annotates the whole line with the corresponding information of that contributor. In this sense, `blame` allows to identify who last modified each line in a given revision of a file, but the information about the origin of the content is unaccounted for. The `blame`-based approachs are a suitable solution to detect defects in collaborative software developement [15] as well as to select experts developers for implementing required changes in programs [12], yet does not provide a suitable mechanism to trace the *first* author of the content at a more fine-grained level such as tokens.

To detect authorship information in Wikipedia article text, several analysis and visualization approaches have been employed. *HistoryFlow* by Viegas et al. [17] assigns sentences of a text to the editor who created or changed them. It doesn't however acknowledge deleted content that was later reconstructed as being written by the original editor. More importantly, by operating on a sentence level, small changes like spelling mistake corrections lead to wrongly recognizing the correcting editor as the author of the whole sentence.

By tracking how long certain words remain unrevised in an article and which editors wrote those words, *Wikitrust* generates a visual mark-up of trusted and untrusted passages in any Wikipedia article [1, 2, 3].[6] To track authorship, longest matches for all word sequences of the current revision are searched for in the preceding revision and in previously existing, but now deleted word-chunks. In this way, *Wikitrust* can as well detect reintroduced words and assign the original author – an important feature, as "reverts" to formerly existing revisions are commonplace in Wikipedia. The underlying algorithm is, however, a variation of a greedy algorithm [2], known to look for local optimal, which in the case of determining word authorship can lead to grave misinterpretations when word sequences are moved rather than inserted or deleted only [5].

Flöck and Rodchenko [7] introduce an authorship attribution approach for Wikipedia based on a hierarchical graph model of paragraphs and sentences. The precision of the results according to the evaluation lies at 59.2% compared to 48.4% for *Wikitrust*, values that are rather unsatisfactory for productive usage that requires users to place confidence in the computed attributions. The work presented in this paper in fact builds on the theoretical foundations layed by the model developed in [7] and advances them further.

The most relevant related work, by de Alfaro and Shavlovsky [6], proposes an algorithm for attributing authorship to tokens in revisioned content based on the concept of processing content as sequences of neighboring tokens and finding "relevant" matches in the revision history given a parameterized rarity function, for the work in question defined as the length of the sequence. To store the authorship attributions the anotated revision history is remembered by means of a "trie" structure. The algorithm is tested in terms of runtime and the size of a materialization of the results, but not precision. The algorithm takes into account reintroduction of text and can keep track of authorship on a word or smaller token level and therefore conformes exactly to the goals set out for our work.

[5]`http://git-scm.com/`

[6]`http://www.wikitrust.net/`

3. MODELLING REVISIONED CONTENT

The following subsections outline our model for representing revisioned content.

3.1 A model based on observations of real-world revisioned writing

When observing collaborative writing in systems that rely on long-term, incremental refinement by a large group of users, namely Wikipedia and its sister-projects, certain patterns become salient. In the following we list our conclusions from these observations and from studying related literature (e.g., [9, 11, 17]). These assertions build the conceptual foundation for the content model developed in the remainder of Section 3.

The first assessment is that a considerable part of editing activity after the initial basic writing phase consists of moving, deleting and reintroducing content, while in fact not adding much new subject matter per edit. A notable number of edits consists of reintroductions, which are mostly reverts due to vandalism; others reasons for reverts are, e.g., "repairing" articles after a perceived defacing of the text. Moves of text sequences are also a regular sight, where a sentence or paragraph gets shifted to another position in the article without alterations. Another sizable amount of edits is predominantly changing only very small parts of the article per edit, incrementally revising the text. This pattern is occasionally interrupted by a burst of new content; for instance, in case of a recent event or a fundamental re-write or addition. Still, very often, the changes implemented per edit do not span more than one section or paragraph – frequently, they don't even transgress the boundary of a sentence. These assertions point out that keeping track of reused or reappearing content methodically plays an important role when intending to efficiently monitor authorship over large data.

Regarding the conceptual definition of "authorship" and, more importantly, the perception by end-users of this term, the larger context of a token plays a crucial role. The paragraph or the section it is embedded in can be as important for the interpretation of its meaning as its immediate neighbours in a sequence. The same exact string of tokens, even up to the length of a sentence (e.g., a figure of speech), might mean something completely different in one section of a text (e.g., an introduction) than in another segment, as it was potentially introduced somewhere else for a different purpose. It can therefore not necessarily be seen as an exact copy of the same sequence of tokens in another position, entailing the attribution of the same author. Tracking provenance hierarchically, by assigning tokens to a larger enclosing unit (sentences), and linking these to another superordinate element like a paragraph provides a more exact identitification of tokens than mere local contextualization.

3.2 A graph-based model for revisioned content

Based on the assumptions presented, we propose a graph-based model to represent revisioned content, where the content is partitioned into units of discourse in writing, i.e., paragraphs, sentences and words (or more generally, tokens).

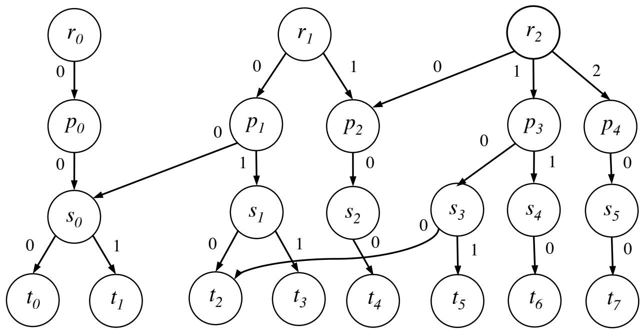


Figure 1: Example of the revisioned content graph. Revisions are represented by nodes r, paragraphs by p, sentences by s, and tokens by t. Arcs between nodes correspond to the *containment* relation.

Definition 1. (A graph-based model for revisioned content). Given a revisioned content document, it can be represented as a k-partite graph, with $k = 4$, $G = (V, E, \mathbb{N}_0, \phi)$ defined as follows:

- The set of vertices V in G is composed of four different subsets of vertices R, P, S, T, i.e., $V = R \cup P \cup S \cup T$. The subset R represents the revisions of a given document (e.g., a Wiki page), P the paragraphs of the page content, S the sentences that compose the paragraphs, and T the tokens (words, special characters, etc.) in the sentences. The subsets of vertices R, P, S, T are pairwise disjoint.

- The set of arcs E in G is partitioned into $k-1$ cuts as follows: $E = \langle R, P\rangle \cup \langle P, S\rangle \cup \langle S, T\rangle$. The arcs in G represent the relationship of *containment*, e.g., if $p \in P$, $s \in S$ and $(p, s) \in E$ then the paragraph p contains the sentence s.

- A labelling mapping $\phi : E \to \mathbb{N}_0$ over the arcs in G represents the *relative position* of a token, sentence or paragraph in a sentence, paragraph or revision, respectively.

In addition, it is necessary to keep record of the sequence in which the revisions were generated. Since adding arcs between revision nodes violates the partite graph definition, we represent this information by annotating the revision nodes with an identifier ($label$) such that if revision r_i is a predecessor of revision r_j, the condition $label(r_i) < label(r_j)$ is met.

In order to illustrate the representation of revisioned content with the proposed model, consider a Wiki page with three revisions r_0, r_1 and r_2 as depicted in Figure 1. The first revision, r_0, contains a single paragraph, p_0, which is composed of only one sentence, s_0, with two tokens (t_0 and t_1). The second revision, r_1, creates two new paragraphs, p_1 and p_2. The paragraph, p_1, is written by reusing the sentence s_0 from revision r_0 followed by a new sentence, s_1. The third revision, r_2, reuses paragraph p_2 from the previous revision and creates two new paragraphs, p_3 and p_4. In addition, $p3$ contains a new sentence which reuses the token t_2 originally inserted in the previous revision.

3.3 Restrictions over the model

In the following we present the restrictions to guarantee consintency within the model. We refer to paragraphs, sentences or tokens as 'content elements'.

The first property refers to the number of content elements within a revision. Particularly, this property allows the definition of an empty revision (with no content elements).

Property 1. The number of arcs that leaves a revision vertex (denoted $deg^+(\cdot)$) must be greater than or equal to 0.

$$\forall v \in R(deg^+(v) \geq 0) \tag{1}$$

The second property restricts the existence of empty paragraphs or sentences, i.e., each paragraph or sentence must contain at least one content element.

Property 2. The number of arcs that leave a paragraph or sentence vertex must be greater than 0.

$$\forall v, v \in P \ \vee \ v \in S(deg^+(v) > 0) \tag{2}$$

The following property establishes that paragraphs, sentences or tokens must be associated to at least one revision, paragraph or sentence, respectively.

Property 3. The number of incoming arcs of a paragraph, sentence or token vertex must be greater than 0.

$$\forall v, v \in P \ \vee \ v \in S \ \vee \ v \in T(deg^-(v) > 0) \tag{3}$$

The last property refers to the labelling of the arcs that leave a given vertex. This property states that each content element (paragraph, sentence or token) can only occupy a single position.

Property 4. The label of an arc that leaves a vertex in R, P or S must be between 0 and the number of arcs that leaves that vertex. The set of arcs that leaves a vertex is denoted as $d^+(\cdot)$. Moreover, the labels of the arcs that leave a given vertex must be unique.

$$\forall v, v \in R \vee v \in P \vee v \in S(\exists e \in d^+(v)(0 \leq \phi(e) < deg^+(v)) \\ \wedge \ \forall e_1, e_2 \in d^+(v)(e_1 \neq e_2 \to \phi(e_1) \neq \phi(e_2)) \tag{4}$$

3.4 Operations over the model

We define four different operations over the model that correspond to the actions that can be performed by editing a document. In the following, $path(a, b)$ is defined as the set of paths from vertex a to vertex b. The first operation defines the creation of a new (initially empty) revision, which consists of adding a vertex to the set of revision vertices.

Definition 2. (Creation of a new revision). Let r_{i-1} be the last revision in the graph. The operation $createRevision(r_i)$ represents the creation of a new revision (denominated *current revision*) and is defined as follows:

$$R := \{r_i\} \cup R \tag{5}$$

The following operation allows creating a new paragraph, sentence or token in a certain position of a given revision, paragraph or sentence, respectively. This operation consists of adding a vertex to the corresponding vertex partition, and an edge in the corresponding arc cut annotated with the position of the element.

Definition 3. (Creation of content). Let x and y be content elements such that y is a new element to be added in x at position α. The operation $createContent(x, y, \alpha)$ is defined as follows:

$$\begin{cases} P := \{y\} \cup P \;\wedge\; \langle R, P\rangle := \{((x, y), \alpha)\} \cup \langle R, P\rangle & \text{if } x \in R \\ S := \{y\} \cup S \;\wedge\; \langle P, S\rangle := \{((x, y), \alpha)\} \cup \langle P, S\rangle & \text{if } x \in P \\ T := \{y\} \cup T \;\wedge\; \langle S, T\rangle := \{((x, y), \alpha)\} \cup \langle S, T\rangle & \text{if } x \in S \end{cases} \tag{6}$$

In addition, the creation of an element y in a given revision r_i meets the following condition:

$$\exists \rho(\rho \in path(r_i, y))) \tag{7}$$

The third operation defines the action of copying, or reintroducing, content from an old revision. This operation consists of creating an arc from the content element of the current revision to the copied element, and annotating the arc with the new position of the element in the current revision.

Definition 4. (Copying content from an old revision). Let r_i $(i > 0)$ be the current revision and r_{i-k} $(0 < k \leq i)$ an old revision. Let x and y be content elements such that y is an element copied from revision r_{i-k} in the element x of revision r_i at position α. The operation $copyContent(x, y, \alpha)$ is defined as follows:

$$\begin{cases} \langle R, P\rangle := \{((x, y), \alpha)\} \cup \langle R, P\rangle & \text{if } x \in R, y \in P \\ \langle P, S\rangle := \{((x, y), \alpha)\} \cup \langle P, S\rangle & \text{if } x \in P, y \in S \\ \langle S, T\rangle := \{((x, y), \alpha)\} \cup \langle S, T\rangle & \text{if } x \in S, y \in T \end{cases} \tag{8}$$

In addition, copying an element y from revision r_{i-k} in revision r_i meets the following condition:

$$\exists \rho(\rho \in path(r_{i-k}, y)) \;\wedge\; \exists \rho'(\rho' \in path(r_i, y)) \tag{9}$$

The last operation is the deletion of content, which models the case when content from the previous revision is removed. This operation requires no alteration on the structures of the model, since elements are never removed from revisioned content.

Definition 5. (Deletion of content). Let r_i $(i > 0)$ be the current revision and y the element from the previous revision to be removed. The deletion of y in r_i meets the following condition:

$$\exists \rho(\rho \in path(r_{i-1}, y)) \;\wedge\; \nexists \rho'(\rho' \in path(r_i, y)) \tag{10}$$

4. AUTHORSHIP ALGORITHM

This section describes the implementation of an authorship attribution algorithm based on the presented content model.

4.1 The authorship attribution problem

The authorship attribution problem consists of identifying for each token the revision in which the token originated. This problem has been previously introduced by de Alfaro and Shavlovsky [6], where each token of a given revision is annotated with an origin label denoted as Θ.

In the following we devise a theoretical solution to the authorship attribution problem for a given token, built on top of the proposed graph-based model.

THEOREM 1. *(A solution to the authorship attribution problem). Let $G = (V, E, \mathbb{N}_0, \phi)$ be the graph of a given revisioned content, modelled according to Definition 1. The authorship of a certain token t can be determined by identifying all the revisions*

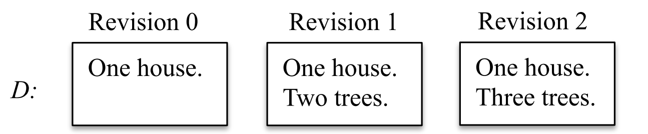


Figure 2: Example of a document D with revisioned content. D contains three revisions, each one with a single paragraph.

where the token occurs and selecting the revison that was generated first in sequential order, i.e., the revision with the minimum label.

$$\forall t \in T(\Theta(t) := \min \{label(r_i) | \exists \rho(\rho \in path(ri, t)) \wedge r_i \in R\})$$

PROOF. We want to demonstrate that if t $(t \in T)$ was originated in revision r_i $(r_i \in R)$, then $\Theta(t) = label(r_i)$. By contradiction, lets assume that $\Theta(t) \neq label(r_i)$. Therefore, we have two cases: $\Theta(t) < label(r_i)$ or $\Theta(t) > label(r_i)$. Furthermore, there exists a revision r_j $(r_j \in R)$ such that $\Theta(t) = label(r_j)$. By hypothesis, t was not originated in r_j but in r_i, then t must have been copied in r_j from r_i. According to Definition 4, an element can only be copied from an old revision, thus the case $\Theta(t) < label(r_i)$ is discarded. In the other case, we can affirm that r_i is a predecessor of r_j, therefore $min(label(r_i), label(r_j)) = label(r_i)$. By the definition of $\Theta(t)$, the only possibilty for not selecting r_i as the origin of t is that there does not exist a path from r_i to t (contradiction to Definition 3). $\square$

4.2 Implementation of the proposed solution

We have demonstrated that with our proposed model it is possible to provide a straightforward solution to the authorship attribution problem in revisioned content. In the following we devise an algorithm to build this model while generating origin labels of tokens simultaneously.

Algorithm 1 outlines our proposed solution, WikiWho, an algorithm that constructs a graph according to Definition 1 to represent a document consisting of revisioned content. WikiWho follows a breadth-first search (BFS) strategy to build the graph structures for each revision.

In order to illustrate the execution of Algorithm 1 consider the revisioned content presented in Figure 2. In this example, the document D is composed of three revisions r_0, r_1 and r_2. The algorithm starts processing r_0, and creates the corresponding revision node (line 4). Then, the content is split into paragraphs; there is only one paragraph (p_0) which is split into a single sentence (s_0). Once the algorithm has tokenized all the sentence nodes, it proceeds to calculate the `diff` operation (line 13) between the current text and token nodes from the previous revision. For the first revision, this operation corresponds to `diff('', 'One house .')`, i.e., empty content diffed vs. the tokens of r_0. The `diff` output states that all the tokens in revision r_0 are new and the algorithm creates the corresponding nodes (line 26), with $\Theta = 0$. The current state of the graph is presented as the leftmost of the three sections of Figure 3.

Following the execution of WikiWho in Figure 2, in the next iteration the algorithm creates the revision node r_1. In this revision the paragraph has changed w.r.t. to the previous revision, therefore the node p_1 is created. One of the sentences of p_1 corresponds to s_0 – created in revision r_0 – and the algorithm marks all the nodes reachable from s_0, including s_0. The other sentence is new, therefore the

Algorithm 1 WikiWho algorithm

Input: A document D with revisioned content $r_0, r_1, ...r_{n-1}$.
Output: A graph $G = (V, E, \mathbb{N}_0, \phi)$ representing the revisioned content from D.

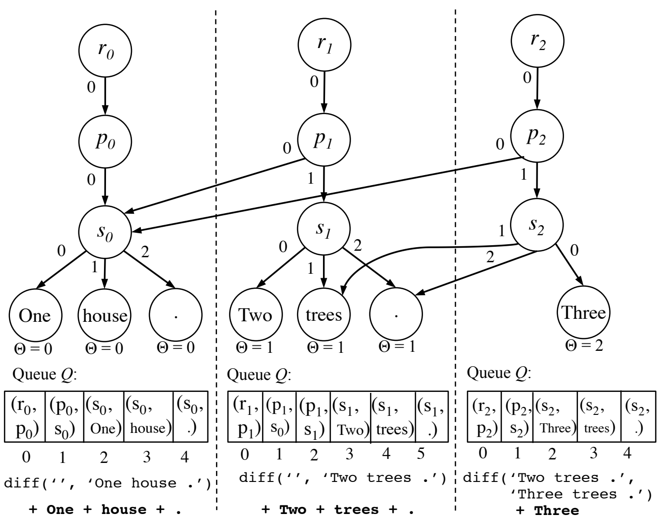
```
1:  create an empty graph $G = (V, E, \mathbb{N}_0, \phi)$
2:  create an empty queue $Q$
3:  for $i$ in $0, 1...n-1$ do
4:      $G.createRevision(r_i)$
5:      $label(r_i) \leftarrow i$
6:      $y' \leftarrow tokenize(r_i)$
7:      enqueue $(r_i, y)$ onto $Q$ for all $y$ in $y'$
8:      $x_{prev} \leftarrow NULL$
9:      $diffed \leftarrow FALSE$
10:     while $Q$ is not empty do
11:         $(x, y) \leftarrow$ Q.dequeue()
12:         if $x$ is a sentence $\wedge$ $!diffed$ then
13:             calculate diff of unmarked tokens of $r_{i-1}$ against unmarked tokens
                of $r_i$ $(i > 0)$
14:             $diffed \leftarrow$ TRUE
15:         end if
16:         if $x = x_{prev}$ then
17:             $\alpha \leftarrow \alpha + 1$
18:         else
19:             $\alpha \leftarrow 0$
20:             $x_{prev} \leftarrow x$
21:         end if
22:         if $y \in V$ $\wedge$ $y$ is not marked then
23:             $G.copyElement(x, y, \alpha)$
24:             mark all the nodes reachable from $y$, including $y$
25:         else
26:             $G.createElement(x, y, \alpha)$
27:             if $y$ is a token then
28:                 $\Theta(y) \leftarrow label(r_i)$
29:             else
30:                 $z' \leftarrow tokenize(y)$
31:                 enqueue $(y, z)$ onto Q for all $z$ in $z'$
32:             end if
33:         end if
34:     end while
35:     unmark all the marked nodes
36: end for
37: return $G$
```

node s_1 is created. At this step, the `diff` is calculated over the sentence `'Two trees .'` and the unmarked token nodes from r_1 (which is $\emptyset$). The three tokens are identified as new and annotated with $\Theta = 1$. The state of the graph at the end of this iteration is illustrated by the combination of the left and the middle section of Figure 3.

In the last iteration of the running example, the node r_2 is created. This revision contains a new paragraph p_2, composed of s_0 and a new sentence s_2. After processing the sentences, the algorithm calculates `diff('Two trees .', 'Three trees .')`. Note that the sentence `'One house .'` is not considered in the `diff` step, since these nodes were marked when the algorithm processed s_0. According to the `diff`, only the token `'Three'` is identified as new, and annotated with $\Theta = 2$. Figure 3 depicts the current state of the graph.

4.2.1 Representation of content nodes

As explained earlier, revision nodes are uniquely annotated with a *label* (line 5 of Algorithm 1). These labels are associated to the sequential order in which the revisions were generated. In a Wiki environment, the revision identifier provided by the system can be applied as *label* in WikiWho. Paragraph and sentence nodes are identified by a hash value. The hash value is the result of applying the MD5 algorithm [16] over the content of the paragraph or sentences, respectively. Token nodes contain the actual text of the revisions, i.e., the words or tokens that compose the revisioned content. WikiWho annotates the token nodes with their corresponding origin label (Θ), as shown on line 28 of Algorithm 1. This avoids the calculation of all the paths from revision nodes to a given token to retrieve the authorsip information.



r0
p0
s0
One
house
.
Θ = 0
Θ = 0
Θ = 0
r1
p1
s1
Two
trees
.
Θ = 1
Θ = 1
Θ = 1
r2
p2
s2
Three
Θ = 2
Queue Q:
(r0, p0) | (p0, s0) | (s0, One) | (s0, house) | (s0, .)
0 1 2 3 4
Queue Q:
(r1, p1) | (p1, s0) | (p1, s1) | (s1, Two) | (s1, trees) | (s1, .)
0 1 2 3 4 5
Queue Q:
(r2, p2) | (p2, s2) | (s2, Three) | (s2, trees) | (s2, .)
0 1 2 3 4

```
diff('', 'One house .')
+ One + house + .

diff('', 'Two trees .')
+ Two + trees + .

diff('Two trees .',
     'Three trees .')
+ Three
```

Figure 3: Execution of WikiWho for the example from Figure 2. Sections delimited by dashed lines represent the state of the graph after each revision. At the bottom, the progress of the queue Q and the output of the `diff` for each revision iteration are depicted.

4.2.2 Implementation of operations

One crucial operation of the presented algorithm is determining whether a certain node y belongs already to the graph (line 22 of Algorithm 1). Depending on the type of the node, this step can be implemented diferently. For instance, if y is a paragraph or a sentence, the algorithm could check only the corresponding node partition, i.e., if $y \in P$ or $y \in S$, respectively. In the case when y is a token, the decision wheter the token is new or not relies on the output of the `diff` operation (line 13).

On lines 6 and 30, Algorithm 1 performs the tokenization of the text unit that is currently being processed. Tokenization refers to the process of splitting the text into more fine-grained units of characters. In this work, we define a token as the smallest unit, be it indivisible. Further details regarding the definition of grammatical units are discussed in Section 4.3.

Once the graph is built, accessing the authorship information for each computed revision is straightforward. The origin labels of the content in r_i can be retreived by exploring the graph G with any search approach, e.g., depth-first search (DFS), considering r_i as the root node and visiting the nodes in the order induced by ϕ.

4.3 Design issue: Tokenization

To achieve authorship attributions that correspond to the "original" author of a given token it is important to chose the tokenization very carefully in respect to the specific context of the system and its usage. An overly fine-grained tokenization, e.g., on character level might prove counterproductive if it is not necessary to determine the origin of single characters and make the interpretation of the results too complex and unusable for end-users. On the other hand, using only demarcations such as periods to identify sentences or even whitespaces to separate what is commonly understood as a "word" can prove to coarse. Whole tokens can in that case spuriously be reattributed to new authors even though only minor readjustments have been applied to the token. As the optimal tokenization can vary immensely between different contexts, we concentrated for this work on a Wiki environment, especially Wikipedia. We believe, however, that the presented design choices are applicable as well for other Wikis that follow roughly the same patterns of editing and collaborative writing.

When processing the complete source text of articles, as we do with the WikiWho algorithm, it is not only important to take into account the intricacies of natural language (as it appears on the article front-end) and a corresponding optimal tokenization but also the much more function-oriented markup language of the Mediawiki software, called "Wiki markup".[7] Small changes in the markup can entail important changes in the front-end of an article, be it for instance the inclusion of a template via only a few pasted characters or the setting of links. Consider the following example, where a contributor writes the word Germany and another contributor in a subsequent revision adds markup content to represent the same word as an internal link ([[Germany]]) to the respective article. Using only white spaces as separators would lead to counting the latter string as a new token; using a similarity-metric like a Levenshtein distance might lead to an attribution of the whole string to the former author. What actually happens here is that the word "Germany" was written by one author and the link was set by another. Both are elementally important and distinct actions in Wikipedia. Many more of these examples could be given, pertaining to infoboxes, templates, language links, references and numerous others.

Besides white spaces we therefore chose the most commonly used functional characters of the Wiki markup as delimiters, such as "|", "[", "]", "=", "<", ">", to name some. We further split sentences (as defined in our model) at common sentence delimiters such as ".", "?" etc., and paragraphs at double line breaks, which are used in Wikipedia to begin a new paragraph in the text. All delimiters were also treated as tokens, as they fulfill important functions in the text. We determined all of these demarcations after extensive testing with real article data until we reached a splitting we deemed optimal to achieve the best balance of precision and effiency.[8]

4.4 Optimization for Wiki environments: Vandalism detection

The most expensive operation of the proposed algorithm is the `diff`. We are interested in detecting those vandalism attacks that change large amounts of content from one revision to another, affecting the performance of the `diff` operation. There are different types of vandalism in Wikipedia, such like removing significant parts of a page, or modifying a page in a way that adds a lot of vandalistic content.[9] This kind of obvious vandalism gets commonly removed in subsequent revisions by another editor or bot. In order to avoid the computation of the `diff` in the previous cases, we implemented two simple vandalism detection techniques that don't impose a significant computational overhead and filter out only the most obstructive cases.

Percentage of size change from one revision to another. This mechanism is triggered when a large amount of content has been removed in a certain revision, by comparing the current content size versus the size of the previous revision. An example of this type of vandalism is *page blanking*, which signifies deleting all the content of a Wiki page.[10] Since the size of the early revisions of a Wiki page can fluctuate notably, this technique is fired when the revisions have reached a certain size. To not filter out revisions where much content is moved to a different Wiki page in good faith, we analyze the edit comment log provided in the article history dumps.

Token density. This proposed technique aims at detecting vandalism that consists of adding large amounts of disruptive content that is often composed of the same text repeated numerous times. For these cases, we propose a measurement denominated *token density* defined as follows. Consider the bag of tokens of a revision r as the result of splitting the revisioned content with a tokenization mechanism. This bag can be formally represented as a multiset T_r, which consists of a set T'_r – constructed from removing duplicates in T_r – and a function $m : T'_r \rightarrow \mathbb{N}_0$ that denotes the number of times an element of T'_r occurs in T_r. We calculate the token density as follows:

$$tokenDensity(T_r) = \begin{cases} \frac{\sum_{t \in T'_r} m(t)}{|T'_r|} & \text{if } T_r \neq \emptyset \\ 0 & \text{if } T_r = \emptyset \end{cases}$$

A high value of token density suggests that added content is composed of the same repeating words. From this computation we discard tokens corresponding to Wiki markup elements, since they may appear several times in an edit and might be misinterpreted as vandalism. These vandalism filters must be configured with very relaxed thresholds such that no false positives are generated, which was successfully achieved with the values set in the experiments of this work (cf. Section 5.2.2). Note that the objective of implementing these techniques is to avoid the computation of the `diff` operation on large amounts of irrelevant content. We do not aim at applying these techniques as reliable solutions for the problem of vandalism detection in Wikis.

5. EXPERIMENTAL STUDY

We empirically analyzed the performance of the proposed algorithm WikiWho and compared it to the algorithm "A3" by de Alfaro and Shavlovsky [6], which can be regarded as the current benchmark for the given task.[11] In our experiments we report on the execution time of the evaluated algorithms as well as their precision in finding the correct revision of origin for a token. Regarding precision, this is the first evaluation for both algorithms. The WikiWho source code, datasets, gold standard and further details of the experimental results are available online.[12] For all articles analyzed in the following evaluations, the full history in XML format was retrieved from the English Wikipedia via the MediaWiki `Special:Export` mechanism.[13]

5.1 Evaluation of precision

In the following we explain the three-step procedure to construct and validate the gold standard. We measured the quality of the evaluated algorithms by comparing their authorship attributions with the results of the gold standard.

[7] `http://en.wikipedia.org/wiki/Help:Wiki_markup`
[8] The list appears in Text.py, part of the algorithm implementation.
[9] `http://en.wikipedia.org/wiki/Wikipedia:Vandalism_types`
[10] `http://en.wikipedia.org/wiki/Wikipedia:Page_blanking`
[11] Retrieved from: `https://sites.google.com/a/ucsc.edu/luca/the-wikipedia-authorship-project`
[12] `http://people.aifb.kit.edu/ffl/wikiwho/`
[13] `http://en.wikipedia.org/wiki/Special:Export`

5.1.1 Creating a gold standard for authorship in revisioned content

To create the gold standard we selected 40 English Wikipedia articles. Ten articles each were randomly picked from the following four revision-size ranges:[14] articles with i) over 10,000 revisions, ii) 5,000-10,000 revisions, iii) 500-5,000 revisions and iv) 100-500 revisions. The reason for this grouped sampling process was to include a sufficient number of articles that present a challenge to the algorithms when picking the correct revisions of origin, as a higher number of revisions naturally increases the difficulty of the task, as more candidate solutions exist.[15] The latest revision at the point of retrieval of the articles was the "starting revision" for whose tokens the authorship was determined and which is denoted in the gold standard. The text plus markup of each of the 40 articles was split into tokens as described in Section 4.3. Out of this tokenized content, for each article, six instances were randomly selected, resulting in a total of 240 tokens. For each of these, the final gold standard contains the revision in which they first appeared (revision of origin) and the starting revision. To assign the correct revision of origin to all of these tokens, we followed three consecutive steps.

STEP 1: Three researchers of the AIFB institute, including the two authors, manually searched the "Revision History"[16] of the respective 40 articles for the origin of each of the 240 tokens in the gold standard independently from each other. No common interpretation of what constitutes a "correct origin" was agreed on beforehand but was entirely up to the individuals. If the researchers initially disagreed on the correct origin of a token, this disagreement could in most cases be resolved. Only in three cases was this not achieved, so that they were excluded from the gold standard and replaced with new randomly selected tokens.

STEP 2: Next, the gold standard was validated by users of the crowdsourcing platform Amazon Mechanical Turk (hereafter called "turkers").[17] We selected two random tokens for each article in the gold standard. We then created a Human Intelligence Task (HIT) on Mechanical Turk for each of these 80 tokens to be validated by 10 distinct turkers each. We paid 15 US$ cents and selected turkers with a past acceptance rate of over 90% and at least 1,000 completed HITs.[18] A HIT was composed of the following elements:[12]

a) A link to a copy of the starting revision of the Wikipedia article with the highlighted token. If the token only appeared in the markup, we represented an excerpt of the markup as a picture next to the front-end text where it appears in the article HTML, explaining to look for it in the markup.

b) A link to the Wikipedia "Difference View" of the revision of origin proposed by the gold standard. It shows which changes the edit introduced that lead to that revision.[19]

c) Detailed instructions explaining how to use the above mentioned pages and a description of what solution was sought. Three different conditions had to be fulfilled by the proposed revision: First, a string equivalent to the token should indeed have been added in that revision (and not only be moved inside the article text). Second, the token added should be the "same" token as highlighted in our gold standard solution. We explicitly left it open to the turkers to interpret what "same" meant to them and gave only one simple, unambiguous example, explaining that not any string matching the gold standard token was looked for but the specific token in the context that it is presented in (e.g. if the token was a specific "and", we would not be looking for any "and"). The third condition was that the token was actually added in the given revision for the first time and not just reintroduced, e.g., in the course of a vandalism revert. Turkers could chose between one answer option indicating a correct revision, three pointing out various errors and a fifth option with a text box if they had found a revision that was more likely be the origin of the token. For option 5, we offered a bonus payment of 5 US$ to propose a better solution than the one we presented and gave a comprehensive manual on how to search the revision history page of a Wikipedia article by hand as well as a list of tools by the Wikimedia community that can be helpful with the task.

RESULTS OF STEP 2: The 800 answers we received as the result of this experiment included 24 answers suggesting a better solution, but none of them fulfilled all three conditions. We therefore reposted these HITs once we assessed them. As these turkers had spent 17 minutes on average for the task and obviously had tried to find a better solution, they were paid bonuses ex-post. Overall, turkers spent from 40 seconds to 13 minutes solving the task, with an average of 4 minutes 49 seconds. Turkers thus spent considerable time assessing the correctness of the presented solutions.[20]

We report on the results aggregating the answers of the "incorrect" options (option 5 was handled as mentioned above). On average, the solutions received 89% agreement. 65 tokens received nine to ten out of ten agreement votes. 12 solutions received 8/10 and three received 7/10 "correct" votes. In the latter cases the disagreeing turkers pointed in 7 of 9 answers at the lack of a matching string being added in the suggested revision, although this was in fact the case.[21] Overall, we consider the result of this experiment to compellingly support the proposed gold standard solutions.

STEP 3: As a further test we ran the WikiWho algorithm as well as the A3 algorithm (in two different variants), as explained in the following Section 5.1.2. For 67 of the 240 tokens in the gold standard at least one of the algorithms produced a result deviating from the gold standard. For all of these 67 tokens we set up a Mechanical Turk experiment with the same settings as explained in step 2. In this HIT, however, we presented the turkers with three differing possible revisions of origin and asked them which one was most likely correct or if none of them was (option 4). One of the three solutions was always the gold standard answer and one or two were solutions by one of the algorithms, depending on how many algorithms results disagreed. If only two differing solutions were available, the third one was filled with an incorrect control answer. Answer positions were randomly changed in each HIT.

RESULTS OF STEP 3: 670 single answers were retrieved for the 67

[14]The "random article" feature of Wikipedia was used. Redirect or disambiguation pages where skipped.

[15]Articles under 100 revisions are not challenging for the task. We did sample test-runs with non-crowdsourced test answers and both algorithms scored very close to a precision of 1.0.

[16]Example for the article "Korea": `https://en.wikipedia.org/w/index.php?title=Korea&action=history`

[17]`http://www.mturk.com`

[18]The pay rate was the result of a number of tries with rates at 10 and 13 cents that did not attract enough turkers.

[19]Example diff: `https://en.wikipedia.org/w/index.php?title=Korea&diff=574837201`

[20]This excludes 12 turkers whose HITs were rejected and reposted for obviously incorrect answers, such as choosing option 5 and not reporting a better solution.

[21]We believe this could have been because of particular nature of the respective diffs, where the token was hard to track.

tokens. The general agreement score to the gold standard solution was 81%, with 7/10 or more votes validating the gold standard as correct for 63 tokens. Given the nature of the task and the different possible interpretations, we consider the gold standard to have gained a solid affirmation for these tokens. In four cases, however, the turkers disagreed decisively with the gold standard. In two of these instances, there was complete disagreement over the right solution, while in two other examples four users each endorsed the differing WikiWho and the differing A3 solution, respectively. We therefore removed these tokens from the following evaluation in 5.1.2.[22] The remaining 63 tokens achieved an agreement of 83%.

As a conclusion to these three steps of quality assurance we can assume that the gold standard is sufficiently robust to test algorithm precision against it. We are however publishing the gold standard and encourage the the community to assess and expand it further.

5.1.2 *Measuring the precision of the algorithms*

After validating the gold standard, the WikiWho and the A3 algorithm were tested for their ability to correctly detect the revisions of origin for each token. The evaluation metric was precision defined as: $p = \frac{TP}{TP+FP}$, where a true positive (TP) means that the authorship label computed by the algorithm is matching the gold standard described in 5.1.1 and otherwise is a false positive (FP).

Three articles in the gold standard from the revisions-size bracket over 10,000 had to be excluded due to technical reasons and are hence exempt from all following experiments to guarantee the same data basis.[23] The remaining 37 articles encompass 218 tokens.

The A3 algorithm we retrieved includes a filter module that seems to be intended to remove the Wiki markup that does not appear on the HTML front-end of an article.[24] More important is however that all citations and references get discarded, although they appear in the front-end and can in some cases make up large parts of the article, not to mention their functional importance for the credibility of Wikipedia articles. This filter thus excludes not only formatting and style elements but also valuable content, whose authorship is of high interest. For that reason we ran one variant of the A3 algorithm with this markup filter disabled (henceforth "A3 MF-OFF"), also because our aim was to compare WikiWho to another algorithm that is able to process the entire source text. The unaltered version of the A3 algorithm will be referred to as "A3 MF-ON".

A3 MF-OFF only yielded results for 138 of the 218 tokens as the remaining part was filtered out. We therefore compared its output to the result for the same 138 tokens by WikiWho, as can be seen in the lower part of Table 1. A3 MF-ON produced output for the whole set and we compared it to the full results of WikiWho, listed in the upper part of Table 1. WikiWho scores at 18% and 15% higher precision overall, respectively for the full and the restricted token sample. As becomes evident from the results, the gain in precision by WikiWho turned out especially high for the two biggest revision-size brackets, while it is lower for the 5000 to 50,000 bracket and much lower and non-existent, respectively, for the smallest size bracket. On one hand, this seems to indicate that for articles with up to 500 revisions, the difference between the two approaches is negligible and both have a very high precision. Given the long tail of small articles in Wikipedia, this is a very encouraging result. On the other hand, with increasing editing activity and therefore growing number of revisions of an article, it seems to become harder for the A3 algorithm to correctly determine the authorship of certain tokens, while WikiWho can sustain a high level of precision, even for articles with over 10,000 revisions. Given the steady growth of Wikipedia and the size of other revisioned content these approaches might be adaptable to, this is an important aspect of scalability. Moreover, particularly when processing the much "dirtier" Wiki markup, WikiWho seems to have a notable advantage when it comes to precisely determining authorship.

[22]We marked these cases in the published gold standard accordingly.
[23]The A3 algorithm did not process these articles despite several intents to resolve the issue. The files were unaltered XML-dumps from the Wikipedia servers. The articles are "Vladimir Putin", "Apollo11" and "Armenian Genocide".
[24]The filter is not described in [6].

Table 1: Precision comparison of WikiWho and A3

$x \in$	ALL	[10k,∞)	[5k,10k)	[500,5k)	[100,500)
		Full sample			
p WikiWho	0.95	0.97	0.93	0.95	0.95
p A3 MF-OFF	0.77	0.77	0.64	0.76	0.87
Gain in *p* by WikiWho	0.18*	0.20*	0.29*	0.19*	0.08
Available results *n*	218	58	42	58	60
		Sample restricted to output of A3 MF-ON (*n* -80)			
p WikiWho (restricted)	0.96	0.97	0.89	1.00	0.95
p A3 MF-ON	0.81	0.69	0.70	0.88	0.95
Gain in *p* by WikiWho	0.15*	0.28*	0.19	0.12*	0.00
Available results *n*	138	39	27	34	38

Notes: *n* = number of tokens, k = one thousand, *p* = precision, x = number of revisions per article, * = difference significant at 0.05 (paired t-test)

5.2 Evaluation of execution time

We measured the algorithm time for computing authorship labelling of revisioned content from Wikipedia pages.

5.2.1 *Experimental set-up*

We used two datasets created by retrieving the full revision history content for each article from the English Wikipedia in XML format.[13] *Dataset 1* was generated by randomly selecting Wiki pages in the article namespace that were no redirects or disambiguation pages. This dataset is comprised of 45,917 revisions in 210 articles, i.e., an average number of 219 revisions per article; the average revision size is 2,968 KB. *Dataset 2* contains the Wiki pages used in the quality evaluation presented in Section 5.1.1. Its articles are larger, with an average number of revisions of 5,952 and an average revision size of 461,522 KB per article. This allowed for some "heavy load" testing. This last dataset is composed of 36 articles with a total of 214,255 revisions.[25]

We defined execution time as the time elapsed between the point when the algorithm reads the first revision of an article and the point in time when the authorship labelling of the last revision of a given page history is computed. Both algorithms are implemented in Python and the time was measured with the `time.time()` command from the Python standard library. The reported experiments were all executed on a dedicated OS X machine with a 2.5 GHz Intel Core i5 processor and 4GB RAM.

[25]We excluded again the three articles mentioned in Section 5.1.2 as well as "Jesus", as it would run over 5 hours for some settings.

5.2.2 Algorithm settings

The A3 algorithm was set according to the configuration presented by de Alfaro and Shavlovsky [6], with rarity function as the sequence length and threshold equals to 4. The tokenization implemented by A3 uses only whitespaces as delimiters. In addition, A3 employs two types of filters. First, a content aging filter that limits the number of revisions to be analyzed by excluding the content from old revisions according to the values of the thresholds Δ_N and Δ_T;[26] in our experiments, we used the original configuration of the algorithm($\Delta_N = 100$, $\Delta_T = 90$). Second is the Wiki markup filter, which we discussed in Subsection 4.3. The Wiki markup affects the amount of content to be processed in each iteration and we thus studied the performance of the algorithm when this filter on (**A3 MF-ON**) and disabled (**A3 MF-OFF**).

Regarding the WikiWho vandalism detection mechanisms presented in Section 4.4, we empirically set up their thresholds by performing tests on a random article sample. In the experiments, the value for the change percentage filter was equal to -0.40, and the token density was set to 10. This resulted in 0.5% of revisions being filtered. As discussed in Section 4.3, the definition of tokenization units is an important factor that affects the quality as well as the performance of the algorithm. We studied the performance of WikiWho in two variations of tokenization plus one additional setting:

- **WikiWho complex tokenization (CT)**: We implemented the tokenization described in Section 4.3, considering the Wiki markup. This is the original algorithm we propose.
- **WikiWho simple tokenization (ST)**: Tokens are obtained by splitting the raw content using only whitespaces as delimiters. This setting was used to assess which additional load the complex tokenization adds by generating a much higher number of tokens to track.
- **WikiWho simple tokenization and content aging filter on (ST/AF-ON)**: We implemented the content aging filter described for A3, with $\Delta_N = 100$. This setting and the A3 MF-OFF configuration allow to compare the algorithms under similar conditions.

5.2.3 Results

We executed each setting 5 times and report on the average time per article, per revision and per Kilobyte. The runtime results for all settings are listed in Table 2. Figure 4 additionally plots the time results in relation to increasing total article history size, meaning average revision size times number of revisions.[27] The figures include the function for a fitted linear line for each setting, showing that for the A3 settings the runtime increases with growing article size by a much larger factor than for the WikiWho variants. Although the WikiWho performance seems to be more volatile, we can observe that the behavior of all the settings in both algorithms is consistent in general and increases in a linear or almost linear fashion with an increasing amount of content. Fluctuations between data points suggest that the execution time is also influenced by other properties of articles, e.g., the amount of content modified from one revision to another.

The runtime decrease by WikiWho in contrast to A3 is in the range of one order of magnitude, differing over the settings. The two most comparable settings ST/AF-ON and A3 MF-OFF differ by a factor of 10.83 and 8.80, respectively for the two datasets, while the originally proposed setting CT completes the task in an even shorter time period. It appears that the time filter is in fact no accelerator for the WikiWho algorithm, supposedly because it creates more overhead than is saved by not processing older revisions. The A3 algorithm shows the same behavior in *Dataset 1*. Still, for *Dataset 2* the markup filter seems to take effect, presumably because in longer revisions the amount of filtered content is larger.

Table 2: Execution time of algorithm settings

Algorithm setting	Avg. time per article (secs.)	Ratio of runtime (base: ST)	Avg. time per revision (secs.)	Avg. time per KB (secs.)
		Dataset 1		
ST	0.84	1:1	0.0027	0.0746
CT	1.04	1:1.24	0.0039	0.0113
ST/AF-ON	1.32	1:1.57	0.0037	0.1069
A3 MF-OFF	14.30	1:17.02	0.0322	1.0280
A3 MF-ON	17.69	1:21.05	0.0408	1.3343
		Dataset 2		
ST	184.97	1:1	0.0184	2.3055
CT	284.44	1:1.54	0.0307	3.6238
ST/AF-ON	290.97	1:1.57	0.0272	3.5191
A3 MF-ON	2559.38	1:13.84	0.2514	32.6031
A3 MF-OFF	2834.37	1:15.32	0.2591	34.8180

Overall, WikiWho is able to execute the given task of labelling tokens with authorship in a very efficient manner and outperforms the A3 algorithm significantly in runtime in all variants. This is possible due to the construction of paragraph and sentence nodes comparable to creating indexes over the text. This allows to efficiently detect large chunks of text that remained unchanged between revisions, reducing the number of comparisons at the token level.

5.3 Evaluation of materialization size

Since revisioned content is in constant production – particularly in the English Wikipedia, where over 3 Million revisions are created monthly[28] – it might be useful to materialize partial computation in order to allow incremental data processing, i.e., the algorithm can be stopped at a certain point in time and then resume its execution. Therefore, we implemented a JSON serialization mechanism to optionally materialize partial computation. We measured the overhead caused by the serialization in terms of space.

5.3.1 Experimental set-up

We used the articles contained in the two datasets presented in Section 5.2, and serialized the computation of the authorship labels of the whole page history for each article. We compared the serialization mechanism of WikiWho and A3 under similar conditions with the settings ST/AF-ON and MF-OFF, respectively. Both algorithms WikiWho and A3 utilize the `cjson` Python library[29] to implement the (de-)serialization mechanisms. Since we are comparing the agorithms with content aging filter with $\Delta_N = 100$, we report on the size of the JSON serialization in relation to the size of the last $N = 100$ revisions of each article.

5.3.2 Results

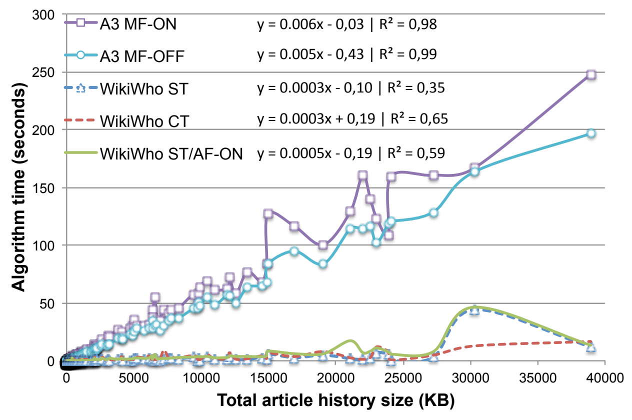
Figure 5 plots the results of the materialization of results for WikiWho and A3. We can observe that the behavior of the two algo-

[26] Δ_N limits the processed content to a maximum of N most recent revisions, while Δ_T further filters out revisions older than T days.

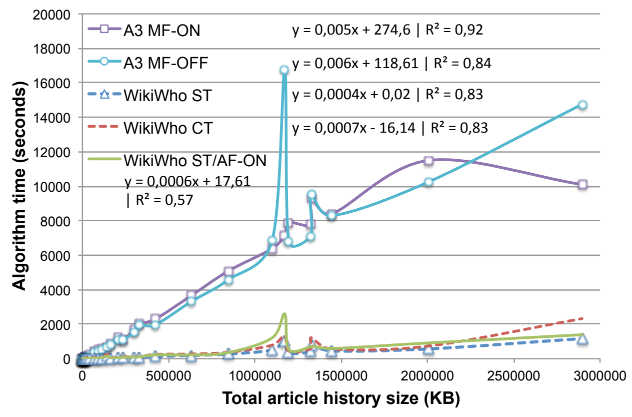
[27] As both values influence the runtime. Text includes Wiki markup.

[28] According to the Wikimedia Statistics of June 2013: `http://stats.wikimedia.org/EN/SummaryEN.htm`

[29] `https://pypi.python.org/pypi/python-cjson`



(a) Performance in *Dataset 1* (Articles randomly selected)



(b) Performance in *Dataset 2* (Articles used in quality evaluation)

Figure 4: Algorithm execution time evaluation for different settings of WikiWho and A3 in *Dataset 1* and *Dataset 2* – *y* denotes the respective fitted linear functions for the data series on the left (fit lines omitted and data points partly omitted for readability)

rithms is in general consistent. When the size of the revisioned content increases, the relative size of the serialization decreases exponentially. This suggests that both algorithms efficiently represent redundant content. On average, the size of the serialization is 0.66 for WikiWho and 0.56 for the A3 algorithm with respect to the total size of the last 100 revisions that were not removed by the content aging filters. Figure 5 further depicts the percentaged overhead difference of the WikiWho minus the A3 materialization with increasing content size. It shows a volatile behavior with a clear linear trend. Weighing the cost of storing the results for small articles versus the average time to calculate authorship labels with WikiWho, materializing these results does not bring additional benefits, on the contrary, it incurs on extra space and time. Therefore, the proposed serialization mechanisms should be executed only when the time to compute the authorship labels over the whole Wiki page history exceeds a "reasonable" response time, e.g., wait time for end users. Using the "worst case" linear estimation from setting CT for *Dataset 2* (cf. Figure 4), a hypothetical maximum runtime of 5 seconds would allow to process all articles with up to 17,300 KB complete revision history text size without the need for materialization. As far as we can estimate by our random sampling and a limited query to the Wikipedia database, at least half of all articles in the English Wikipedia currently stay under this size limit.[30] The needed storage space can of course be further reduced by relaxing the runtime constraint. For articles over this limit, intervalls of revisions can be determined when a materialization becomes necessary, although this is beyond the scope of this paper.

[30]`https://wiki.toolserver.org/view/Database_access`

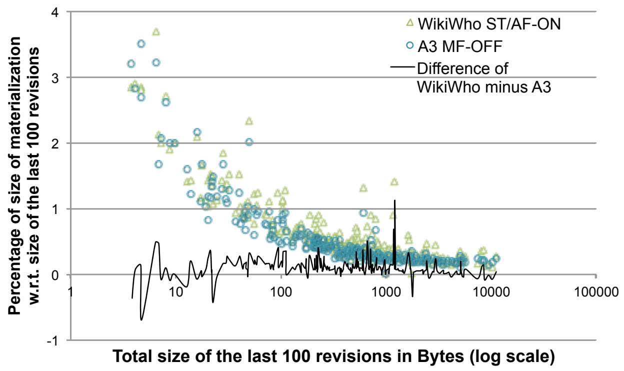


Figure 5: Performance in *Dataset 1* (Wiki pages randomly selected) – article "Rankin County" at y-axis values 10.69 (WikiWho) and 11.13 (A3) not shown for readability

6. CONCLUSIONS AND FUTURE WORK

In this work we have proposed WikiWho, a solution for the attribution of authorship in revisioned content. We built a model to represent revisioned content based on graph theory, and provided a formal solution to the authorship problem. In order to measure the quality of the proposed solution, we created a gold standard over 240 tokens from Wikipedia articles, and corroborated it via crowdsourcing mechanisms. It is, to our expertise, the first gold standard of this kind to measure the precision of authorship attributions on token-level by an algorithm. We compared WikiWho against the state-of-the-art, exceeding it by over 10% on average in precision, and outperforming the baseline execution time by one order of magnitude. Our experimental study confirmed that WikiWho is an effective and efficient solution.

Although we restricted in this paper the use of WikiWho to single articles, it is also possible to operate it over several or all articles in a Wiki, tracking the movement of text between different pages. Moreover, we used the example of the English Wikipedia as inspiration and testing ground, yet the proposed solution can be understood as a specfic implementation of a more generally applicable method for revisioned content. We are convinced that many of the assumptions made for the use case of the English Wikipedia also hold true not only for other language editions and Wikis but also for other revisioned content systems.

Future Work: We plan to study further techniques to optimize the materialization of intermediate computation. Since each article may show different editing patterns (in terms of size and number of revisions), it is benefical to adapt the frequency of the serialization routine for each article in an optimal way. We will also look at compression mechanisms that allow to reduce the size of the materialization. Currently we are establishing an API for WikiWho to provide the possibility to query the authorship annotation for arbitrary revisions in the English and German Wikipedia.

Acknowledgements

We thank Andriy Rodchenko and Felix Leif Keppmann for their contributions to this research. This work was partially supported by grants from the European Union's 7th Framework Programme under grant agreement number 257790 (FP7-ICT-2009-5).

7. REFERENCES

[1] B.T. Adler, L. de Alfaro, I. Pye, and V. Raman. Measuring author contributions to the wikipedia. In *Proceedings of the 4th International Symposium on Wikis*, WikiSym '08, pages 15:1–15:10, New York, NY, USA, 2008. ACM.

[2] T. Adler and L. Alfaro. A content-driven reputation system for the Wikipedia. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 261–270, 2007.

[3] T. Adler, K. Chatterjee, L. Alfaro, M. Faella, I. Pye, and V. Raman. Assigning trust to Wikipedia content. In *International Symposium on Wikis*, 2008.

[4] R. Baggen, J. Pedro Correia, K. Schill, and J. Visser. Standardized code quality benchmarking for improving software maintainability. *Software Quality Journal*, 20(2):287–307, 2012.

[5] R.C. Burns and D.D.E. Long. A linear time, constant space differencing algorithm. In *Performance, Computing, and Communications Conference, 1997. IPCCC 1997., IEEE International*, pages 429–436. IEEE, 1997.

[6] L. de Alfaro and M. Shavlovsky. Attributing authorship of revisioned content. In *Proceedings of the 22nd international conference on World Wide Web*, WWW '13, pages 343–354, Republic and Canton of Geneva, Switzerland, 2013. International World Wide Web Conferences Steering Committee.

[7] F. Flöck and A. Rodchenko. Whose article is it anyway?–detecting authorship distribution in wikipedia articles over time with wikigini. In *Online proceedings of the Wikipedia Academy 2012*. Wikimedia, 2012.

[8] A. Forte and A. Bruckman. Why do people write for wikipedia? incentives to contribute to open-content publishing. group 05 workshop position paper. In *GROUP 05 Workshop: Sustaining Community: The Role and Design of Incentive Mechanisms in Online Systems. Sanibel Island, FL*, pages 6–9, 2005.

[9] J. Jones. Patterns of revision in online writing a study of wikipedia's featured articles. *Written Communication*, 25(2):262–289, 2008.

[10] F. L. Keppmann, F. Flöck, A. Adam, E. Simperl, D. Rusu, G. Holz, and A. Metzger. A knowledge diversity dashboard for wikipedia. In *Proceedings of the ACM WebSci'12*, New York, NY, USA, Juni 2012. ACM.

[11] A. Kittur, B. Suh, B. A. Pendleton, and E. H. Chi. He says, she says: conflict and coordination in Wikipedia. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '07, pages 453–462, New York, NY, USA, 2007. ACM.

[12] M. Linares-Vasquez, K. Hossen, H. Dang, H. Kagdi, M. Gethers, and D. Poshyvanyk. Triaging incoming change requests: Bug or commit history, or code authorship? In *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, pages 451–460, 2012.

[13] C. M. Pilato, B. Collins-Sussman, and B. W. Fitzpatrick. *Version control with subversion*. O'Reilly Media, Inc., 2009.

[14] C. R. Prause. Maintaining fine-grained code metadata regardless of moving, copying and merging. In *Source Code Analysis and Manipulation, 2009. SCAM '09. Ninth IEEE International Working Conference on*, pages 109–118, 2009.

[15] F. Rahman and P. Devanbu. Ownership, experience and defects: a fine-grained study of authorship. In *33rd International Conference on Software Engineering (ICSE)*, pages 491–500, 2011.

[16] R. L. Rivest et al. Rfc 1321: The md5 message-digest algorithm, 1992.

[17] F. B. Viégas, M. Wattenberg, and K. Dave. Studying cooperation and conflict between authors with history flow visualizations. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '04, pages 575–582, New York, NY, USA, 2004. ACM.