UNIVERSITY OF CALIFORNIA

SANTA CRUZ

WIKITRUST: CONTENT-DRIVEN REPUTATION FOR THE WIKIPEDIA

A dissertation submitted in partial satisfaction
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

B. Thomas Adler

June 2012

The Dissertation of B. Thomas Adler
is approved:

Professor Luca de Alfaro, Chair

Professor Scott Brandt

Professor Neoklis Polyzotis

Tyrus Miller
Vice Provost and Dean of Graduate Studies

Contents

List of Figures

List of Tables

Abstract

WikiTrust: Content-Driven Reputation for the Wikipedia

B. Thomas Adler

The Wikipedia was initially created to promote collaboration between writers before submitting their work to a peer review process, to address complaints about the speed of peer review. Ironically, the criticism most widely levied against the Wikipedia is the lack of accountability for authors, and the potential to misinform readers. There is a large community around the Wikipedia project which actively fixes errors as they are discovered, but an unending stream of vandals and spammers chip away at the good will of volunteers who maintain the project for the collective good. We suggest that vandalism detection systems can be used to help direct the volunteer effort on changes more likely to be a problem, making more efficient use of the project's human resources.

We use edit distance to quantify the effort of authors, and propose automated methods to evaluate the quality of this effort and how they might be combined into an author reputation system. We desire that an author's reputation be correlated with the stability of the text they contribute — low reputation should be a predictor of future author contributions being edited or deleted. Reputation can then be another input to a vandalism detection system.

Instead of measuring the "truth" of contributions, our quality ideas measure the "group consensus" in a piece of text. As the article text stabilizes over time, we conclude that it has reached a form which most members of the community can reasonably agree on. As group collaboration increases in prominence on the Internet, we feel that this research will open the door on new applications and quality measures.

To Pokey and Lala,

who both taught me what I really needed to learn.

♡ ♡ ♡

Acknowledgements

Thanks to Luca, who never gave up when I needed him and stuck with me through the ups and downs of the years. And to my labmates and co-authors, Marco, Vishwa, Ian, Pritam, Leandro, Krish, and Axel: your moral (and research) support was invaluable at so many points for keeping me engaged. Karen provided endless edits to my grammar and diction, often to my chagrin. Thanks to each of you: I learned a great deal and appreciate all that you shared with me.

There are a few heroes that saw me through the darkest hours; I am indebted to you for your caring when mine failed me. And there are so many friends and family, new and old, that were characters in this adventure and gave me their support and encouragement and compassion. You all have my deepest thanks.

I am the luckiest one.

The text of this dissertation includes excerpts of previously published material; copyright of this material remains with its respective holders and appears here with their permission. Chapters 3, 4, and 6 expand on the initial paper presenting our content-driven reputation ideas for the Wikipedia [2]. Chapter 5 is a reprint of our investigation into contribution measures [5]. Chapter 7 covers similar ground as (and includes some material from) two previously published works [4, 3]. Illustrations from PhD comics are copyright Jorge Cham [16], with many thanks.

This work was supported in part by the Center for Information Technology Research in the Interest of Society (CITRIS), and by the Institute for Scalable Scientific Data Management (ISSDM).

Chapter 1

Introduction

1.1 World Wide Collaboration

The dot-com boom of the late 1990's brought the open source movement into mainstream consciousness, bringing with it the mantra "information wants to be free" [115]. In the midst of this environment, the Wikipedia[1] first appeared. The Wikipedia is an online encyclopedia using an open model of group collaboration where anyone can contribute: when an article is displayed, any reader can click on an "edit" button to modify the text as they see fit. Thanks to this openness, the Wikipedia has grown to over 3.4 million articles and as of September 2011, is the seventh most visited site on the web[2]. This large-scale group collaboration has become known as *crowdsourcing*. By encouraging visitors to contribute their own content, sites adopting this model[3] hope to grow rapidly as users build on each others' work [103].

The particularly open model of group collaboration that the Wikipedia embodies

[1] `http://www.wikipedia.org`

[2] According to Alexa traffic rankings, `http://www.alexa.com`

[3] For example, Flickr, YouTube, StackExchange and even Facebook.

in allowing anonymous contributions also receives much criticism about the potential for misinformation [94, 93, 56, 46, 25, 99, 90, 97], both intentional and accidental. A study comparing the quality of the Wikipedia against that of the Encyclopædia Brittanica found that the number of errors in the Wikipedia is very near that of the curated work [36], but this achievement is not attained by mere chance. As an online encyclopedia, accuracy of information is a major concern, thus the Wikipedia community has developed a process of eternal vigilance around screening edits: their volunteer *RC Patrol* scans all recent changes and reverts edits that they do not consider suitable [117]. The important feature of the Wikipedia that enables this process is that all past versions are kept for each article. Users can easily roll back an article to a previous version, undoing the contributions of other users. A fundamental insight behind wiki development is that, if well-intentioned and careful users outnumber ill-intentioned or careless users in the community, then valuable content will predominate, since the undesired contributions are easily undone [57].

Online communities have a typical lifecycle: a small community develops and rallies around unifying principles; then the community grows and attracts a more diverse group of members; finally, the relative anonymity of a large community encourages a small "anti-social element." The problem for the Wikipedia is how to keep these bad actors at bay. Obvious vandalism is easy to identify and revert, but minor changes to factual information can be quite insidious. For example, changing a date by a few days is difficult for anyone to verify as a correction and not vandalism. Most famously, an anonymous user created a new biography entry within Wikipedia with the following text [94, 93, 92]:

> "John Seigenthaler Sr. was the assistant to Attorney General Robert Kennedy in the early 1960s. For a short time, he was thought to have been directly in-

volved in the Kennedy assassinations of both John, and his brother, Bobby. Nothing was ever proven."

This mixture of fact and fiction is very plausible given the mythic nature of the Kennedy assassination, and it is impossible for casual readers to validate without checking reference materials.

1.2 Related Work

As the Wikipedia becomes a standard resource for the internet public, there is rising interest in quality measures. A series of incidents shows that the Wikipedia can be manipulated, despite the "many eyes" reviewing the site: a prank biography [94, 93, 92]; congressional aides adjusting political biographies [56, 46, 25]; a user pretending to be a professor [97]; and a slew of other self-interested parties making inappropriate edits [11, 39, 71]. Articles have been written questioning the general credibility of the Wikipedia [99, 90], and a scientific study addressing the question has been published [36]. Should schoolchildren be allowed to use the Wikipedia as a resource when they might encounter misinformation or foul language at any time [38, 73]?

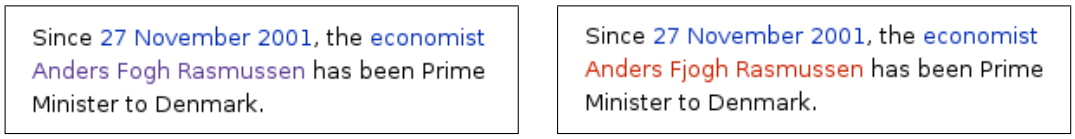
Today, several lines of research are pursuing vandalism detection specific to the Wikipedia [80]. These solutions all apply machine-learning techniques to annotated data sets, treating the task as a "supervised learning" problem. When we started our research, there were no annotated data sets available. We chose instead to track the work of identified users and to compute a trust value for the text created, which we passively reveal to readers by coloring the background of untrusted text. Another advantage of tracking the actions of individuals is to correlate those actions into a signal which might reveal bad actors.

The idea of assigning trust to specific sections of text of the Wikipedia articles as a guide to readers has been previously proposed in the scientific literature [64, 22, 126], as well as in white papers [52]; these papers also contain the idea of using text background color to visualize trust values. There is also a report on whether such a visualization is useful to readers of the Wikipedia [62]; this report uses manually generated colorings which ignores some of the subtleties specifically included in WikiTrust to reduce the problem of habituating users to coloring [2].

Other studies of the quality of the Wikipedia have focused on trust as article-level, rather than word-level, information. These studies can be used to answer the question of whether an article is of good quality, or reliable overall, but cannot be used to locate the portions of text within an article that deserve more scrutiny, as our work is able to [1]. In Zeng et al. [127], which inspired [126], the revision history of a Wikipedia article is used to compute a trust value for the entire article. In [31, 65], features derived via natural language processing are used to classify articles according to their quality. In [59], the number of edits and unique editors are used to estimate article quality. The use of revert times for quality estimation has been proposed in [107], where a visualization of the Wikipedia editing process as a *history flow of text* is presented; an approach based on edit frequency and dynamics is discussed in [125]. A fast-growing body of literature reports on statistical studies of the evolution of Wikipedia content, including [107, 108, 74]; we refer to [74] for an insightful overview of this line of work.

1.3 The Problem

The Wikipedia's continued success depends on ill-intentioned users not being able to overwhelm the well-intentioned users. Communities are complicated systems, with people constantly joining and leaving their membership. We can suppose that people



(a) Before the modification [113]. (b) Immediately after the modification [114].

Figure 1.1: An attempt to modify the spelling of the Danish Prime Minister's last name, from Fogh to Fjogh. The casual user, who does not speak Danish and happens to check the version history, has little indication on which might be correct. This is a case of vandalism (in Danish, a *fjog* is a *fool*) that is quite subtle and most users would not even notice.

continue to join the Wikipedia community because most pages are substantially useful, and users feel that their contributions *add* to the resource, giving them a sense of satisfaction [9]. If the amount of vandalism occurring were to increase so that most users were only *repairing* the Wikipedia, there would be much less satisfaction.

The key problem this community faces is in quickly identifying vandalism to prevent the appearance (to casual users) of needing maintenance. The RC Patrol [117] acts as a guardian, investing a large amount of human capital to scan all edits made to the Wikipedia, searching for vandalism and other inappropriate edits. Still, it is possible to sneak vandalism by the RC Patrol; the article on South Pasadena, California was vandalized in May 2008 to include a Nazi propaganda film, which persisted until April 2009.[4]

Vandalism can be very hard to identify for inexperienced users. Consider the example of Figure 1.1, which is an excerpt from the Politics of Denmark entry in the English Wikipedia: an anonymous user substitutes "Fjogh" for "Fogh" in the Prime Minister's last name. This is a particularly subtle kind of vandalism, because it requires knowledge of Danish (*fjog* translates to *fool* or *goofy*) to recognize this change as anything more than a spelling correction. A sophisticated reader might recognize

[4]See Chapter 4 for more details.

the broken link as a hint on the true spelling, or even try to use Google to research the name (both spellings return results, however). The level of effort to recognize the error and to verify this relatively small detail is quite high.

Beyond vandalism, another issue that affects the perceived quality of the Wikipedia are well-intentioned users that contribute very low-quality material. For instance, material written from a neutral point of view[5] promotes the Wikipedia as an impartial resource that welcomes contributions from all parties. When a contribution does not adhere to this standard, the bias detracts from the credibility of the article and can discourage other users from participating in the community.

Historically, there are three inter-related approaches to attaching value to knowledge. The most familiar method is *formal publication* by an organization that stakes their reputation on the material; the Encyclopædia Britannica is one famous example. To achieve their high quality, experts write the articles in the Encyclopædia Britannica and then peers and an editorial staff review them. *Peer review* is a second approach to the creation of knowledge; science extensively uses this method, but we distinguish it from "formal publication" because only the actual authors stand behind the veracity of the material being reported. *Reputation* is a third approach attaching value to knowledge; people naturally ascribe reputations to other individuals and will accept knowledge without verification from sources that they feel have a high enough reputation.

The overarching question for the Wikipedia is *how do we maintain the good quality of articles?* The historical approaches have all translated to online knowledge repositories (e.g., Google Knol[6] and Encyclopædia Britannica are based on formal publica-

[5] `http://en.wikipedia.org/wiki/Wikipedia:Neutral_point_of_view`

[6] `http://knol.google.com`

tion, Nupedia [116] was based on peer-review, and Stack Exchange[7] implements an explicit reputation system). We propose that constructing an automated reputation system for the Wikipedia would facilitate the detection of vandalism.

1.4 Contributions of this Work

This dissertation takes up the question of how to help readers of the Wikipedia understand the quality of an article. Our approach is to examine the revision history of an article and summarize the evolution of text through a reputation system rating the authors of the text.

We develop a quality measure to reflect the reaction of the community to an edit. This quality measure is consistent with the less general "reverts are bad" notion used in other works [95, 48, 8, 112], but is more nuanced in allowing a contribution to be revised and still be considered as adding some value to the project. As part of constructing this quality measure, we consider the author attribution problem for collaborative works with a revision history, and a definition for edit distance applicable to measuring contributions in the collaborative work.

Using the edit quality measure as a basis, we show how to construct a *reputation system* for authors. We demonstrate that this reputation system has the useful property of predicting the edit quality of future edits by the same author. With the availability of annotated corpora for the vandalism detection problem, we also develop a machine learning solution based on our WikiTrust technologies. We extract features based on calculations made as part of the reputation system and show that the resulting predictions perform competitively against other vandalism detection solutions.

[7] `http://stackexchange.com`

The programs resulting from this research are open source, available under the BSD license from our project website: `http://www.wikitrust.net/`.

1.5 Outline

We start by defining some terminology and notation in Chapter 2 that will be used in multiple later chapters. We then introduce the author attribution problem in Chapter 3. To accord credit to authors for the work they do, it is first necessary to identify their contribution properly. We extend a greedy text difference algorithm [85, 13] to account for the existence of multiple authors and the possibility of restoring text from older revisions.

Chapter 4 then looks at measuring the size of an author edit. The well-known answer is to use edit distance [58], but we discover that the traditional formulation must be adjusted to take into account the rearranging of text that some editors contribute. We refine the definition to build on the result computed by the algorithm in Chapter 3, and account for the rearranging and substitution of text. Finally, we conclude the foundational elements by developing a measure of edit quality. We construct this measure to estimate the approval or disapproval of later authors on the same article. By aggregating the judgement of multiple later authors, we arrive at a value that represents the community's assessment of an edit.

In Chapter 5 we propose a simple model to combine quality measures into a measure of the positive value of an author's *contribution*. Chapter 6 constructs a reputation system out of the contribution measure, and evaluates the performance via several different metrics. Chapter 7 uses the reputation system as a feature for machine learning in the vandalism detection problem, and compares the performance to other solutions for the same problem.

In Chapter 8, we present some directions for future work, both within the specific confines of the Wikipedia and a more general view of reputation as a form of personalization.

Chapter 2

Definitions

The following terminology and notation is used throughout this work.

2.1 General Notation

$\mathbb{A}$ – the set of main Wikipedia *articles*. We use this term interchangeably with the term *pages*, as each article appears as a single web page on the Wikipedia site. For this work, we only consider articles that are in the NS0 (Main) namespace to be in $\mathbb{A}$. Talk and User pages, for example, are in other namespaces.

$\mathbb{U}$ – the set of registered Wikipedia *users*, plus a single anonymous user. MediaWiki stores IP address information for anonymous users, but due to the ambiguity of tracking users this way, we map all anonymous use to the single anonymous user.

$\mathbb{V}[a]$ – each article $a \in \mathbb{A}$, has a history of versions describing how that article has evolved over time. We denote the chronologically ordered list of all $n > 0$

revisions of article a by

$$\mathbb{V}[a] = v_1, v_2, \ldots, v_n \tag{2.1}$$

Version v_1 is the first instantiation of the article, and version v_n is the most recent. A common pattern among Wikipedia editors is to save checkpoints of their edits, so that there are several consecutive revisions by the same author. We *filter* revisions to keep only the last in a sequence of consecutive revisions by the same author. Thus, throughout this work, we assume that two consecutive revisions of an article never have the same author.

$\mathbb{V}$ – the chronologically ordered list of all revisions, across all articles in $\mathbb{A}$. That is,

$$\mathbb{V} = \mathrm{sort}_{\mathrm{time}} \left[\bigcup_{a \in \mathbb{A}} \mathbb{V}[a] \right] \tag{2.2}$$

2.2 Chapter 3 – Author Attribution

$\mathrm{Words}(v)$ – gives the sequence of words that make up the content of version v; see Definition 3.1.

$\mathrm{RevAuthor}(v)$ – every revision of an article has a user associated with the edit. This function gives the user associated with v of an article. For anonymous users (who are distinguished by IP address in MediaWiki), we map each to the single anonymous user in $\mathbb{U}$. (Definition 3.2.)

$\mathrm{PrevRevs}(v)$ – for $v \in \mathbb{V}[a]$, this is the ordered list of revisions previous to v in $\mathbb{V}[a]$; see Definition 3.3.

$\mathrm{BestMatch}(v_i, r, \mathrm{PrevRevs}(v_i))$ – a function that locates the best matching text for the r^{th} word of v_i in the text of previous revisions. For further description and

development, see Equation 3.4 and the accompanying text.

$d_a(i, j)$ – is the edit distance between revisions v_i and v_j of article $a \in \mathbb{A}$. Chapter 3 explores several possible definitions for edit distance.

$TSurv_a(i, j)$ – for article $a \in \mathbb{A}$, measures the amount of text introduced in revision v_i that survives to v_j.

2.3 Chapter 4 – Contribution Quality

$q^n_{tdecay}(k)$ – the text decay quality of revision v_k (we leave the article to which v_k belongs implicit in the context of usage), as given by the the n revisions after v_k. The value of this quality is given by the solution to Equation 4.2.

$J^n_a(i)$ – a function that returns the set of *judging* revisions associated with revision v_i of article $a \in \mathbb{A}$. A judging revision for revision v_i is a revision v_j such that $0 < j - i \leq n$ and $\mathrm{RevAuthor}(v_i) \neq \mathrm{RevAuthor}(v_j)$. Thus, the set is limited to, at most, n filtered revisions which are nearest to v_i in time. See Definition 4.6.

$q^{a,n}_{elong}(k)$ – the edit longevity quality of revision $v_k \in \mathbb{V}[a]$, as judged by up to n judging revisions after v_k. See Definition 4.7.

2.4 Chapter 5 – Sizing Up Authors

$\mathrm{RevPos}(v_i)$ – the revisions of an article form a chronological sequence; this function gives back the position of the revision in the sequence of revisions for the single article.

Chapter 3

Author Attribution

3.1 Introduction

The Wikipedia is both a collaborative work and a versioned document; it is uniquely the largest such work in colloquial language. Programming projects (especially open source projects) are also collaborative and versioned — which makes for an interesting analogy because many of the same questions can be asked:

1. Who contributed how much to the project?
2. How much do the different components interact?
3. How do the components evolve over time?
4. How many components are there in the project?

This chapter expands on ideas originally described in [2].

Of these questions, we focus on determining who contributed how much to the project; as a user contributes more, they gain experience in the community mores, and our working assumption is that their future contributions will be more valuable. That is, our first principle was to give credit to users for the words that they write: more words would lead to more credit.

In the field of software engineering, the question of "who contributed how much" is generally answered by the measure *Source Lines of Code* (SLOC).[1] Computing who added which lines of text is achieved by applying a basic text difference algorithm [68, 105, 13] to two consecutive revisions. The output of the algorithm allows us to determine which lines were added and deleted at each edit, and to attribute the insertions as contributions by the author of the second revision. In principle, the same technique works in the case of tracking authorship on the Wikipedia and should be easy to implement.

There are a few problems with this initial model. The most obvious is that *lines* are too coarse grained a unit for tracking the evolution of English prose; editors frequently come in and revise phrases or even specific words, so it is necessary to compute differences at a finer granularity; we chose to work with white-space delimited *words*. A bigger problem is that text is not only inserted or deleted; it can also be copied or moved around. Variations on text difference algorithms can detect moved and copied text, but to whom should we attribute this text: the author who created the original text, or the editor who rearranged it into its final form?

To answer this last question, let us consider a fairly extreme example. Suppose user Alice creates a new article and writes a page of text for it. Vanessa is a vandal and decides to *blank* the article, i.e., she deletes all the text for the article, but not the article

[1] `http://en.wikipedia.org/wiki/Source_lines_of_code`

itself. Robert is part of the RCPatrol, and he notices that the article has been tampered with, so he restores all the text back to Alice's version. Now who should receive credit for writing the text: Robert or Alice?

We call this the *author attribution* problem for revision collaborative works; and in the context of the Wikipedia it seems obvious that Robert's work is valuable maintenance, but that Alice is the true "author" of the text. Thus, a better model of attribution (than just giving credit to the person who inserted the text most recently) would instead give credit to the author who originally created the text. For example, the WikiTravel[2] site creates a tree representing the version history of an article: two consecutive versions have a parent-child relationship in the tree, except when the second version is identical to an earlier version; versions that are identical are merged into a single node in the tree, which is attributed to the author of the earliest such version [82] (see [30, 87] for similar variations on organizing revisions). Using this revision tree, the WikiTravel site computes the authors of the article as being the set of authors starting from the most recent version and following parent links to the root of the tree.[3]

Given this framework, a second example is now easier to analyze for attribution:

> Since 27 November 2001, the economist Anders Fogh Rasmussen has been Prime Minister to Denmark [113]. *As Prime Minister to Denmark, the economist Anders Fogh Rasmussen leads the government with the consent of Queen Margrethe II.*

Alice wrote the first sentence of this example, and Robert added the italicized text, but Robert's content repeats much of the same information as in Alice's sentence. Clearly,

[2] `http://wikitravel.org`

[3] Note that WikiTravel computes a set of authors as a requirement of the license agreement that applies to contributed content, so the problem is more than an academic one.

a model of attribution which simply tracks insertions and deletions is too simple, as Robert would receive credit for words that are not his own.

We present a method for computing text authorship based on first computing a difference between multiple previous revisions and the target revision. From our two examples, we see already that the differencing algorithm must allow matching over multiple past revisions, as well as supporting multiple copies of the same block of text. Computing these differences in the context of the Wikipedia requires efficiency to avoid unwieldy CPU requirements or execution times of several months; to this end, we resort to a method based on greedy differencing algorithms [85, 13]. Using a greedy algorithm for computing the difference, we are able to find matches of old text and propagate authorship information to the target revision fast enough to maintain a trust annotated copy of the Wikipedia in real-time on a single workstation.

3.2 Related Work

Traditionally, the *author attribution* problem is one of identifying the anonymous author of a work, based on characteristics of writing style and word choice [50]. Our context is different: we analyze a revisioned collaborative document, where the author of each revision is known. The difficulty in our situation is determining how to assign "credit" for words when they might be a reintroduction of older text. To approach this problem, we consider difference algorithms as a starting point.

Finding the difference between two strings is a long-studied problem known as *string-to-string correction* [110]. Initial work in computer science on this problem revolved around finding the Longest Common Subsequence (LCS) [42], which identifies

the sequence of symbols[4] in common between the two strings. From there, it is easy to identify the optimal set of insertions, deletions, and replacements to transform one string to the other. The well-known UNIX `diff` utility [47] is based on this algorithm. Myers shows that LCS and shortest edit script are equivalent to finding shortest/longest paths in an edit graph [68].

There are a score of variations on the basic problem of comparing strings; an excellent survey of the field is presented in [89]. An important concept emphasized in [89] is that difference and distance analyses generally take one of three forms: *trace*, *alignment* and *listing*. Traces are used to represent the common units (e.g., letters or words) between the source and target strings.

```
B   I   R   D
|       |       \
B   O   R   E   D
```

Alignments also find the commonality between the source and target strings, but allow more flexibility in the specification of the non-common portions.

```
B   –   I   R   –   D          B   I   –   R   –   D
B   O   –   R   E   D          B   –   O   R   E   D
```

Listings are the most general of all, being organized as a sequence of elementary edit operations to transform the source string into the target string.

```
BIRD      Delete I
BRD       Insert O
BORD      Insert E
BORED
```

[4]Note that the symbols need not be contiguous.

The original string-to-string correction problem statement only permitted insertion, deletion, and substitution operations, but other applications (for instance, spelling correction) require allowing *transposition* of characters to be an available operation. Lowrance and Wagner [61] tackle this natural extension, and produce an algorithm to solve it using *restricted traces* which allow simple transpositions. Wagner later develops another algorithm which allows unrestricted traces, essentially calculating the listing distance [109, 89].

Our solution for WikiTrust greedily selects the *best match* (according to some criteria we define later) from the set of all possible matches between the set of strings, marks the match, and then proceeds to find the next available *best match*. The general principle is the same as that used in the Smith-Waterman algorithm [96] (also [89, Ch. 10]) for *local sequence alignment*, which iteratively locates the best local alignment between two nucleotide sequences.[5] Our solution is a specialization that does not allow insertions and deletions within a *best match* (in the context of Smith-Waterman, the score for an insertion or deletion would be $-\infty$). While we forbid insertions and deletions to be components of a *best match*, we still have other preferences on matching: we prefer matches to be similarly situated in their strings. To achieve this, we modify the overall score of a proposed match to take into account the position of the match in the source and target strings.

Concerned with storing deltas as part of a revision control system, Tichy investigates the idea of finding the shortest edit script as the primary goal in solving the string-to-string correction problem. As part of achieving that goal, Tichy introduces *block moves* that describe a section of text in the source string as exactly matching

[5]Nucleotide sequences are more commonly referred to as DNA sequences, represented as sequences of A, C, T, and G to represent the nucleotide base pairs. Matching between two sequences of DNA is important for understanding, among other things, the similarities and differences between species.

a section of text in the target string [105]. This is the same notion as *transpositions* by Lowrance and Wagner [61], but Tichy is optimizing for shortest edit script where there is no penalty for transpositions of blocks of characters. (It should also be noted that "shortest" specifically depends on the encoding of a block as a single edit operation.) Tichy's algorithm loops through the target string and greedily chooses the longest match from the source string, which Tichy proves as generating the shortest edit script transforming the source to the target string. This greedy solution is refined by subsequent work in several ways: indexing on string prefixes [72], efficient generation of deletion operations [85], and restricting block moves to be in sequence [13]. Our work evolved from this line of research, but incorporates the notion of *best match* in a way not dissimilar to that found in the Smith-Waterman algorithm [96].

Our work presents a different dimension of the string-to-string correction problem. In previous formulations, solutions are optimized for abstract performance characteristics (e.g., running time or edit distance [23, 58]); these solutions sometimes result in edit scripts which are confusing to human readers. This confusion arises from "unnatural" edit scripts that ignore boundaries at the sentence or paragraph level to achieve efficiency, in contrast to how humans think of content at multiple levels of abstraction. For WikiTrust, our interest in string-to-string correction is in trying to estimate the amount of effort put forth by editors, so we prefer edit scripts which are more likely to describe the actions taken by human editors rather than those which are most efficient. To achieve this, we use the block moves of Tichy [105], but rather than using greedy selection of the longest match given a specific starting location, we perform a global greedy selection of the *best match* (e.g., the longest match) anywhere within the source and target strings, reminiscent of the Smith-Waterman algorithm [96]. Fong and Biuk-Aghai extend the WikiTrust work by applying the hierarchical differencing

idea of Neuwirth et al. [70] to our differencing algorithm, and additionally classify components of the edit script according to common behaviors of Wikipedia editors.

Historians of computer science will note a relation to *transclusions* [69]. Nelson's vision for hypertext included the notion of micropayments to authors, which required detailed and manual attributions of text. We propose automatically detecting attribution (equivalently, transclusions of portions from earlier revisions) in the context of a revisioned document edited by multiple authors.

3.3 Tracking Text Authorship

To answer the question of which author contributed what text to a collaborative document we consider the problem, without loss of generality, for a single article $a \in \mathbb{A}$, with $n > 0$ revisions given by

$$\mathbb{V}[a] = [v_1, v_2, \ldots, v_n].$$

We define the content of version v_i as being a sequence of $l_i \geq 0$ words for all $0 < i \leq n$, given by:

$$\text{Words}(v_i) = [w_1, w_2, \ldots, w_{l_i}]. \tag{3.1}$$

For us, a *word* is a whitespace-delimited sequence of characters in the Wiki markup language: we work at the level of such markup language, rather than at the level of the HTML produced by the wiki engine. We also desire a function that gives the *author* of a revision; for a user $u \in \mathbb{U}$ that edited and committed article a when version v_{i-1} was the previous version which user u edited to create v_i, we define:

$$\text{RevAuthor}(v_i) = u. \tag{3.2}$$

The MediaWiki software associates user u to version v_i in the database (or the IP address, in the case of anonymous users), so that this information is readily available.

We conceive the problem of author attribution as a recursive relation: the attribution of words in some version of the document depends on the attribution of matching words from earlier versions. At an abstract level, we can discover the inductive step by examining the first several versions. Clearly, for v_1, we have the author of the edit, $\mathrm{RevAuthor}(v_1)$, as the author of each individual word. To track the authorship of words in v_2, there are two cases:

1. A sequence of words in v_2 also exists in v_1. In this case, we retain the original authorship of the words, $\mathrm{RevAuthor}(v_1)$.

2. A sequence of words in v_2 does not also exist in v_1. In this case, the sequence must have been inserted by $\mathrm{RevAuthor}(v_2)$, and we assign authorship accordingly.

Word authorship in v_3 is similar to the situation in v_2, with an additional case:

1. A sequence of words in v_3 also exists in v_2. In this case, we retain the original authorship of the words that was determined for v_2.

2. A sequence of words in v_3 does not also exist in v_2, but does exist in v_1. Again, we retain the original authorship of the words, as it was determined for v_1.

3. A sequence of words in v_3 does not exist in any previous revision. This sequence must have been inserted by $\mathrm{RevAuthor}(v_3)$.

The general flavor of the computation is now clear, but to describe it more precisely, we need some additional definitions. For a given revision, we need to know the

ordered list of revisions earlier than v_i:

$$\mathrm{PrevRevs}(v_i) = [v_j \colon 0 < j < i] \tag{3.3}$$

And for some particular word w_r which is the r^{th} word of $\mathrm{Words}(v_i)$, we need a function which will return the location of the so-called "best match" from a list of earlier revisions:

$$\mathrm{BestMatch}(v_i, r, \mathrm{PrevRevs}(v_i)) = \begin{cases} (v_k, s) & \text{for a best match } w_s \text{ occurring in } v_k. \\ \emptyset & \text{if there is no match.} \end{cases} \tag{3.4}$$

where $v_k \in \mathrm{PrevRevs}(v_i)$ since we are only interested in matches with earlier revisions; we will better describe this calculation in the next section.

This definition of $\mathrm{BestMatch}()$ is extremely general in that it only defines the inputs and outputs of the function; it merely says that word w_r of v_i matches with some other word w_s of v_k, where v_k is in the past of v_i. There is no restriction, in this definition, that w_s be the earliest (or latest) match in the history of revisions, or that it even be the exact same word (as in the case of misspellings). The details of the strategy to rank matches and determine the best match are application dependent, and you can imagine many refinements to how matches are made. For example, consider the word "score": in isolation it is fairly unmemorable, but as part of the quotation "four score and seven years ago" it becomes a signature that allows nearly any American schoolchild to identify the author. We felt that identifying this "context" for a word was important to correctly finding matches, so WikiTrust gives preference to matching contiguous sequences of text. This is described in more detail in the next section.

We can use the same idea of a recursive relation to define the author of each word

$w_j \in \text{Words}(v_i)$:

$$\text{TxtAuthor}(v_i, j) = \begin{cases} \text{TxtAuthor}(v_k, s), & \text{if BestMatch}(v_i, j, \text{PrevRevs}(v_i)) \\ & \quad = (v_k, s) \\ \text{RevAuthor}(v_i), & \text{if there is no best match text.} \end{cases} \tag{3.5}$$

3.4 Matching Text

Our overall goal is to track *text evolution*, by which we mean observing how text shifts around and is added to or deleted from — understanding how it is changed from version to version by examining the differences. This is exactly the output from algorithms solving the string-to-string correction problem.

The simplest string-to-string correction algorithms note only insertions and deletions, but we are also interested in noting whether text has been copied (or even just moved) to another part of the article so that we can assign authorship correctly. We think several properties are desirable when determining how text is reorganized:

- When text is duplicated within an article, we prefer to assign authorship to the author of the original copy. This prevents a vandal from duplicating text and then deleting the original copy. To achieve this goal, we must use an algorithm that allows block moves of text and allows the same source text to be matched multiple times in the target revision.

- If the same text is found multiple times in both the previous and new revision, there are multiple ways to describe how the text has evolved. We would like to give preference to the most plausible explanation: that the text was not rear-

ranged. To do so, we prefer to match chunks of text in the same relative order in their respective document versions.[6]

- We do not want to over-reward the first author to use common words (e.g., "the," "of").

- When text is deleted in one version and then restored in a later version, we prefer the original author. This prevents edit wars or vandalism from disrupting the authorship of text.

- When looking for matches, we prefer to match against chunks of text which have been *live* most recently. That is, when matching against text from multiple revisions, we prefer to match against the most recent revision that is a good fit. This property accounts for editors who review the recent history of an article and restore text that was deleted, as often happens when vandalism occurs.

- We prefer longer matches.

As described in Section 3.2, there is extensive literature on matching text between two strings. Existing algorithms use somewhat different specifications than we have outlined above, generally optimizing for the size of the edit script or for the longest matches, and always only from a single source string to the target string. To achieve our goals, we develop a variation of the greedy algorithms by modifying the greedy step; instead of selecting a match based on length, we use a notion of *match quality*.

The procedure for our greedy-based differencing algorithm[7] is:

---

[6]This might seem irrelevant, given that duplicated text is given the same author. This property becomes important when computing the edit distance from one version to another, where text in the same relative position does not increase the edit distance. If the difference reflected block moves that crossed over each other, then there might be a positive edit distance contributed by the swapping of equivalent text, depending on the definition of edit distance used.

[7]Appendix A lists a Perl language implementation of this basic algorithm.

1. Compute all possible matches (of every length):

$$\begin{aligned}\mathrm{Matches}(v_a, v_b) = \{(i,j,k) \quad | \quad & \exists i \in \mathbb{N}, \exists j \in \mathbb{N}, \exists k \in \mathbb{N} \,.\, (0 \leq f < k, \\ & \mathrm{Words}(v_a) = [p_1, p_2, \ldots, p_{l_a}], \\ & \mathrm{Words}(v_b) = [q_1, q_2, \ldots, q_{l_b}] \,. \\ & p_{i+f} = q_{j+f})\}.\end{aligned}$$

2. For each match, compute a quality score and insert the match into a priority queue, so that the highest quality matches will be drawn first.

3. Draw a match out of the priority queue. If any part of the match has already been previously matched in the target string, then discard this match. Otherwise, record the match as a block **Move** from the source string to the target string. Repeat this step until the priority queue is empty.

4. Check for all unmatched blocks of text in the source string and record them as **Delete** operations in the final edit script.

5. Check for all unmatched blocks of text in the target string and record them as **Insert** operations in the final edit script.

Building our list of matches (as given by $\mathrm{Matches}$) roughly follows the work of other greedy differencing algorithms [85, 13]: build a hashtable of string prefixes that stores the list of locations where each prefix can be found in the source string. Then consider each position of the target string(s) by finding the list of matching string prefixes from the hashtable. For each match, find the extent of the match (beyond the length of the string prefix) and add every matching substring into the priority queue as a separate match.

The key to matches meeting the list of criteria we have defined at the beginning of this section is in the quality function.[8] To prefer matches where the blocks are in similar positions, we can select a quality function which compares the relative positions of a match. Consider a source string of length l_1 and a target string of length l_2, with a match occuring between them of length k at position i_1 in the source and i_2 in the target; then we can define a quality to preserve the ordering of blocks as:

$$q_{block} = -\left|\frac{i_1 + k/2}{l_1} - \frac{i_2 + k/2}{l_2}\right| \tag{3.6}$$

This formula computes the midpoint of each match and compares the relative positions within the full strings; q_{block} will be zero when the matches are in the same relative position and decreases as the blocks move away from each other.

More important than matching blocks not crossing each other, we prefer to match longer pieces of text. To achieve this, we use a tuple to represent the quality. Given that a match has a length of k, we can represent preferring longer matches over non-crossing blocks as (k, q_{block}). The use of a tuple for the priority reflects that we prefer matches of longer length, before considering the issue of match quality.

So far, this discussion has applied equally to the task of computing a difference between two strings and the problem of computing text authorship. In the latter, we actually need to compute matches with multiple previous revisions to account for text that was deleted and then later restored. We describe these previous revisions as *chunks* of text, and number them so that chunk 0 is the most recent revision. As stated before, we prefer matches with more recent revisions than with older revisions; if we let c be the

[8]Except for multiple matching of source blocks, which cannot be handled by the quality function. The source in Appendix A enables this feature by a flag.

chunk number that a match comes from, one possible quality tuple is $(-c, k, q_{block})$.[9]

Once we have an edit script that defines how text evolves from previous revisions to the target (current) revision, we can then propagate authorship from the previous revisions to the target. A simple procedure for this would be:

1. Assign authorship of all words in the target to the current author.
2. Compute the edit script describing the text evolution from previous revisions.
3. For each block move in the edit script, extract the authorship from the source chunk and propagate it to the target.

Optimizations

The Wikipedia is a huge corpus of documents, and processing speed is a crucial factor in doing timely analysis. We can take a few steps to reduce the size of the computation we have described so far. Some optimizations that we have implemented for the differencing step are:

min words We can reduce the number of potential matches examined by the algorithm by requiring a minimum number of words to match before a string will be added to the priority queue. WikiTrust accomplishes this by indexing word tuples in the hashtable of matches computed for the greedy algorithm; for edit distances, we index word pairs, and for text authorship we track word triples.

This works out well, because we prefer not to reward authors that are the first to insert very common words or expressions into an article. The ideal solution

[9]There are other possible quality tuples (e.g., $(k, -c, q_{block})$, if the absolute longest match is preferred), and understanding the ramifications of choosing different quality functions we leave to future research.

would use language analysis to determine n-gram frequencies and compute a score (e.g., tf-idf [49]) for words and phrases.

max matches The hashtable constructed for the greedy algorithm is used to find the initial string prefixes for all possible matches. If there are too many matching string prefixes (for example, a common phrase such as "to the" might appear many times within a single article), then we believe that the string prefix is too common to really be considered to have been authored by a single individual. WikiTrust ignores string prefixes which have more than 50 matches.

longest match Put only the longest possible match onto the priority queue, instead of every possible match (that is, don't place substrings of the longest match onto the priority queue). This saves a lot of CPU and memory by not having to store these substrings in the priority queue. Also, since the longest match is likely to be the one that is actually selected, there is a savings from not having to remove the substrings from the priority queue later. The complication of this optimization is that if a string on the priority queue has been partly matched by an earlier selection, then the residual non-matched parts of the string need to be placed into the priority queue for possible selection later. This makes the assumption that the quality function always prefers longer matches over shorter matches, so that we encounter matches from the priority queue in the correct order. Appendix B demonstrates this modification.

prev matches An optimization that we later implemented on top of **longest match** is **prev matches**: if a potential starting position for matched text is part of a longer match, then the previous position would have been in the set of matches returned by the index of matching positions. That is, if we are examining position j in the

target string and are considering match **Move**(i,j), we know that it was part of a longer match if $i-1$ is one of the matches returned for position $j-1$. In this case, we do not need to examine the match.

header/trailer When users edit an article, the beginning and end of the article are not likely to change much. If the first few words of an article are the same from one revision to the next, it is reasonable to conclude that they are a match without having to test other possible sources of block moves from the article; this fits well with our desire that the resulting edit script try to match a human description of the edit. This pre-matching of the heading and trailing portions of the article can significantly reduce the number of potential matches that are computed in the initial step of the algorithm. Some care should be taken in the handling of authorship; if the header or trailer is duplicated elsewhere in the article, then the original authorship still needs to be retained.

To give an idea of the value of the optimizations, we implemented some variations of the text differencing algorithms and measured the execution time on differencing of the initial filtered revisions of the article Santa Cruz Beach Boardwalk. We used the well-known Tichy algorithm with its own optimizations [105, 72, 85] as a baseline, and called this **diff1**. The most basic implementation of the WikiTrust algorithm we called **diff9**; it places *every* potential match (including substrings of longer matches) on a priority queue before processing for matches, and implements the **header/trailer** optimization. For **diff8**, we instead incorporate the **longest match** optimization which places only the longest matching sections (that is, no substrings of longer matches are considered) onto the priority queue. Our fourth variation is **diff5**, which includes the **header/trailer**, **prev matches**, and **longest match** optimizations. The numbering of these algorithms is to remain consistent with the evaluation done in Chapter 4; see

Source Rev	Target Rev	Num Words	Tichy **diff1**	WikiTrust **diff5**	WikiTrust **diff8**	WikiTrust **diff9**
8741260	12175065	587	128µs	514µs	808µs	526,492µs
12175065	14057051	589	125µs	120µs	791µs	133µs
14057051	17039312	588	122µs	127µs	893µs	131µs
17039312	20060015	588	117µs	123µs	1111µs	132µs
20060015	20551181	588	129µs	153µs	1053µs	136µs

Table 3.1: The execution times of four text differencing algorithms implemented in OCaml, three of which are variations of the WikiTrust algoritm, on the initial filtered revisions of the article Santa Cruz Beach Boardwalk. The particularly slow difference for **diff9** is typical for cases where both the beginning and end of an article are edited, defeating the **header/trailer** optimization; in this case, an image was added near the beginning and a category was added to the end.

Table 4.2 for a breakdown of algorithm variations in that context. See Appendix E for the specific implementation details of each algorithm; some representative run times are shown in Table 3.1. The difference in run times between **diff8** and **diff9** reveal that the **header/trailer** optimization is the most significant in reducing the run time; notice that **diff1** (the Tichy-based algorithm) performs extremely well without the **header/trailer** optimization, but also doesn't really benefit from including it (revealed by timings in Table 4.8).[10]

There are additional optimizations we can make for computing text authorship, as well. A very simple step is to reduce the size of the list of previous revisions given by $\mathrm{PrevRevs}(v_i)$. As defined in Section 3.3, $\mathrm{PrevRevs}(v_i)$ is the ordered list of all previous revisions, $[v_1, v_2, \ldots, v_{i-1}]$. We use the full list of previous revisions to ensure that restored text is assigned to the right author. In practice, the history of an article can extend to several thousand revisions, but an editor is likely only to look back a small number of revisions when searching for text to restore. We arbitrarily limit the list of

[10]The Tichy algorithm matches the initial "header" of the target revision as its first step, so the **header/trailer** optimization does not significantly reduce the work performed by the algorithm.

previous revisions to ten:

$$\mathrm{PrevRevs}(v_i) = [v_j \colon max(1, i-10) \leq j < i]\,,$$

but observe that better selections can be made, as described in [17].

We can further speed up tracking authorship by reducing the number of potential matches, similar to the optimization for the differencing algorithm. If we assume that the quality function always prefers matches from more recent revisions to matches from older revisions (as is the case with $(-c, k, q_{block})$ as a quality function), then we can construct the edit script piecewise by differencing only a single revision at a time. When looking at older chunks, the fact that the target has already been partially matched will reject many potential matches. Appendix C shows an implementation of the basic text tracking algorithm, and Appendix D shows our optimization; Table 3.2 shows representative running times.

An additional optimization we can make is again with respect to the handling of the previous revisions. In defining the matching algorithm, we give preference to more recent versions before falling back to matches against older revisions. Let us consider the case where a piece of text does not match in the most recent revision v_k, but does match in some earlier revision v_i, so that we have $i < k$. Since the text does not match in any v_j for $i < j \leq k$, the text must have been deleted in v_{i+1} and never restored. The size of deleted text in any revision is typically much smaller than the size of the live text, so our final optimization is to check for matches against the deleted text rather than the full live text of previous revisions. Note that this can change the size of matches that are found if a piece of text only partially matches deleted text, so that the quality function might not select matches in a preferred way.

Target Rev	Previous Revs	WikiTrust BasicTextTracking	WikiTrust FasterTextTracking
12175065	1	12.50s	13.93s
14057051	2	36.52s	2.49s
17039312	3	68.79s	3.61s
20060015	4	144.42s	3.14s
20551181	5	933.44s	21.50s
...	...	...	...
24358877	10	5064.09s	26.94s
25306944	10	6717.15s	33.74s
34009105	10	6480.42s	3.14s

Table 3.2: Comparing the execution times of our basic and fast text tracking algorithms, both written in Perl, on selected versions of the article Santa Cruz Beach Boardwalk. Note that Perl is an interpreted language, slower than the OCaml implementation evaluated in Table 3.1 and Chapter 4. We present this data to give a flavor for the importance of the optimizations. The faster text tracking algorithm reduces amount of work done by only matching against one revision at a time, eliminating potential matches found in other revisions.

Thoughts On Evaluation

Historically, text differencing algorithms are judged on the complexity of the algorithm, or how the memory usage scales with the size of the inputs, or in the size of the resulting edit script. These are important concerns, especially in the context of trying to process the over 1.5TB of data that make up the revision history of the English Wikipedia. Our goal is more than efficiency, however; we are trying to measure the work done by authors of a collaborative work. People are far from efficient, so our solution attempts to simply model a human conception of how text is rearranged and reinstated from previous revisions.

The problem we are faced with, then, is how to evaluate this measure; do our calculations correlate well with human intuition? Is it even possible to survey a large number of people to generate edit scripts describing the transformation from one revi-

sion to the next, or can edit scripts only be generated by video taping editors caught in the act? Although we do not evaluate the performance of our difference algorithm directly for this metric, we speculate that it might be possible to indirectly evaluate the performance by examining and evaluating a different problem which uses a difference algorithm as a basis. We investigate this idea in the evaluation of our edit quality measures, presented in the next chapter.

3.5 Conclusions

The problem of tracking text authorship across multiple revisions is not well-studied. We have proposed several criteria to define preferable matches, but other possible formulations might be equally valid.

A significant problem in defining the criteria for a good solution to tracking authorship is the handling of short matches. For example, consider the two-word match "of the," where adjacent words are not part of the same match. Most people would describe the two words as completely unoriginal (to be discarded via a stop-word list [53], or possibly always to be part of some larger phrase), so that a match so small should always be discarded. Incorporating the n-gram frequency into the match quality score would be one way to avoid giving away credit for phrases commonly used as a unit in the language.

Now consider the case of a larger n-gram: suppose there is a match of the phrase "the President of the United States" between a new revision and some older revision. We have just suggested that statistically improbable phrases are good candidates for ascribing authorship to the first person to introduce the phrase into an article. For example, in an article about Caltech, the first person to add text describing a commencement speech by "former President of the United States, Bill Clinton" should certainly get

credit for any appearance of the phrase "President of the United States" in later revisions. What about when this phrase appears in an article titled "President of the United States?" If we apply the same rule, the first person to introduce the phrase might receive much credit if the phrase is used many times throughout the article. Clearly, there is some threshold frequency within a single article where a phrase is no longer *original*.

The challenge illustrated by these two issues is that the notion of authorship really revolves around both ideas and specific words reflecting those ideas. Until natural language understanding makes more progress, this seems like a problem that heuristics will have to address (e.g., as is done in tf-idf [49]). How does one compare two different text tracking algorithms? How would we discover other cases that demonstrate where our specifications of the solution are incomplete? These are open problems outside the scope of this work.

Our own work has chosen to balance a faster implementation with the need to model a human view of text differencing, and in this chapter we have presented some basic algorithms to solve text differencing and author tracking in the context of an edit history. After including several optimizations, we have achieved run times of about a week to process several years of article history for the English Wikiedia.

Chapter 4

Contribution Quality

4.1 Introduction

One of the key problems in trying to build a reputation system for the Wikipedia is that, while a massive amount of data is available, there is little information about how well different users or articles are performing. Having such data is important because we would like one aspect of the reputation system to be descriptive of the community behavior seen in practice. To describe an edit as bad or good, we have to know what the community thinks of it.

One obvious way to do this is to examine reverts in the article history. A revert undoes the action of one or more edits, usually leaving the article in a state exactly matching an older version. For example, if user Alice vandalizes an article by blanking it, user Bob can revert her changes by restoring the article to the state before Alice's edit. If there are constructive edits after a bad edit, it is also possible to selectively undo

This chapter expands on ideas originally described in [2].

just the bad edits. When a revert happens, it is a clear indication that some member of the community believes that the reverted edit is completely inappropriate, so many researchers use this as an indicator of community feedback [2, 95, 48, 8].

There are two issues with using reverts and undos as measures of community feedback. The first is that there are no definitive annotations[1] indicating that reverts or undos have occurred, but computation can detect them.[2] The second, more significant issue is that reverts and undos are a very blunt feedback mechanism; they indicate only complete disapproval. There are no gradations in this evaluation, which does not make it a good measure for the quality of a revision, except for judging the very worst edits.

When we started the WikiTrust project in 2006, our initial thought was to use page views to get a measure of how well reviewed an article is. Since the Wikipedia works due to the collective action of everybody reading the articles, we expected that pages that receive a lot of views would be more accurate because of the higher collective amount of scrutiny received; the more eyes looking over the text, the better reviewed the article would be. (At the time, we abandoned this line of thinking because page view information was not available. Since then, Priedhorsky et al. show how page views can be estimated from some available data sets [81]. Log files are also now available[3] and have a user-friendly front-end[4] for simple queries.) Our experiences during the course of this research is even bleaker than that: plentiful shallow review is not equivalent to deeper review, and more eyes does not mean that anyone will take the effort to correct a mistake. For example, the South Pasadena, California page was vandalized in May

[1] When using the MediaWiki software, a standard notation appears in edit comments, but this is not 100% reliable, as reverts can be effected manually.

[2] Detecting undos can be fairly expensive, so it is typical to restrict checking to a limited number of older revisions.

[3] `http://dammit.lt/wikistats/`

[4] `http://stats.grok.se/`

2008[5] to add the film "Triumph of the Will" as being filmed in that city; it was not corrected until April 2009.[6] This is remarkable because the Nazi propaganda film is well known in film studies, and South Pasadena is in the Los Angeles area, where film studies is very popular. We know from the data available now that roughly 75 people a day read this article, so we have to wonder at how this error might have persisted for nearly a year.

The data from the history of revisions seemed to be our only data source for community sentiment, so we began to examine edits and attempt manually to track how each edit fared in its future. Although there is a great deal of variation in edits, this examination led us to the hypothesis that text contributions might follow a pattern of exponential decay. That is, if an edit is not very good, the bulk of it will be removed right away, with small amounts more being removed in subsequent edits until some kernel of the original edit stabilizes and becomes a fixture within the article. This became our *text quality* measure, explored further in Section 4.3.

Of course, not all useful work consists of adding text to the Wikipedia. The RC Patrol and other maintainers all do work that involves deleting text or editing and rearranging text contributed by others, and any measure that only looks at how text is added will not capture it. How does one go about measuring how much of a delete is preserved in future revisions? We could answer this by treating it as the complement of insertion (that is, counting words restored from the delete as a penalty to the original delete), but measuring word rearrangement was a completely different beast. The answer came in thinking of the evolution of an article like a drunken walk along a road

[5] http://en.wikipedia.org/w/index.php?title=South_Pasadena,_California&oldid=211466067

[6] http://en.wikipedia.org/w/index.php?title=South_Pasadena,_California&oldid=282177714

from one village to the next: there is a definite start and a definite end, but the steps along the way do not always form a neat path and might even drift off the road for a time. Defining a measure of forward progress led to our second quality measure, *edit quality*, which we explain in Section 4.4.

Our motivation for the edit quality measure is that the future versions of an article represent a consensus by the community about which contributions were useful and which were not. Each later author is implicitly making a commentary about the existing text when they decide to make an edit. This sort of inference is very like the idea of *revealed preferences* in economics [88, 106], in which consumer preference is inferred from data about their actual purchases. Similar to the economic setting, a valid criticism of our inference is that later authors do not necessarily review the entire article or its recent history, and so might not be making decisions about all past contributions. We believe that modeling user attention — so that there is greater belief in our inference for text near the edits made, and less belief for portions of the article not modified — can improve our confidence in the inference made. We leave the exploration of this idea to future work.

The analogy to software engineering raised in Chapter 3 suggests a host of potential quality measures, such as understandability, completeness, and reliability [121]. The difficulty of these measures is that they cannot be measured in a programmatic way. (As an example, the reliability of the Wikipedia was compared to that of the Encyclopædia Brittanica by employing experts to review hand-selected articles from both reference works [36].)

4.2 Related Work

Some of the literature studying Wikipedia uses the notion of edit quality as a basis for some other research goal, and so they use the gross measure of detecting reverts to signal a poor quality edit [2, 95, 48, 8]. The WikiTrust project introduces the idea of text and edit longevities [2], which are finer grained than a binary classification and are the subject of this chapter. Similar to the notion of text longevity is the idea of Persistent Word Revision per word [43, 44] (PWRpW), which counts the number of revisions that added words survive and normalizes that sum by the number of words added (that is, it computes the average number of revisions that added words survive). PWRpW differs from text longevity in that it is essentially computing the area under the curve of text survival (such as that depicted by Figure 4.1), whereas text longevity is trying to model the shape of the curve (see Figure 4.2).

At a broader view, quantifying the quality of an edit is strongly related to the problem of detecting vandalism. Many machine learning models produce a probability that an edit should be classified as vandalism, and this probability can be directly taken as a quality score. We refer the interested reader to Section 7.2 for a discussion of the literature around this view of the problem.

4.3 Text Quality

When first approaching the topic of measuring the quality of contributions made by users, our thinking (shaped by some of the public discussion surrounding contributions [102]) focused on the text being added by users. The basic assumption of a reputation system is that past performance is a reliable indicator of future performance, so we were asking the question "does text added by some users *survive* longer than text

by other users?" Note that there are other possible measures of quality, for example: what size is the contribution, what is the reading level of the text [34, 41], how good is the grammar, and does the text seem to be related to the topic of the article [48]. We decided not to tackle these kinds of quality measures, both for the difficulty of natural language processing and because such methods would require significant work for each language we wanted to support. Measuring survival has the advantage that it is relatively cheap to compute and works the same way across most languages.

To calculate how long a piece of text survives, we need to track the authorship of units of text[7], and then compute authorship again in later versions of the article to find those words with an authorship dating back to the revision we are trying to determine the quality of. In Chapter 3, we describe how to track text authorship for the text in an article $a \in \mathbb{A}$. Just tracking authorship is not enough for tracking survival, since we need to know what specific revision a word came from. Let us define another recursive relation, like for $\mathrm{TxtAuthor}()$ in Definition 3.5, which defines the revision where a word was first introduced. For a word w_j, the j^{th} word of $\mathrm{Words}(v_i)$ (where $v_i \in \mathbb{V}[a]$), we define:

$$\mathrm{TxtSrcRev}(v_i, j) = \begin{cases} \mathrm{TxtSrcRev}(v_k, s), & \text{if } \mathrm{BestMatch}(v_i, j, \mathrm{PrevRevs}(v_i)) \\ & \quad = (v_k, s) \\ i, & \text{if there is no best match text.} \end{cases}$$

Now we can define the text survival of words to be the number of words introduced in

[7]For the WikiTrust project, we opted to use a granularity of words to reduce computational requirements.

v_m still present in v_n:

$$TSurv_a(m,n) = |\{j : \exists j \in \mathbb{Z} \,.\, (\mathrm{TxtSrcRev}(v_n, j) = m)\}| \tag{4.1}$$

Text Decay Quality

In the abstract, our goal is to define some measure that we can compute for revisions that quantifies an estimate of the *quality* of the revision. There are many possible ways to define quality measures, which is the subject of research on vandalism detection (see Section 7.2 for background on that topic). As an example, a simple quality measure would be the heuristic that inappropriate words added as part of an edit would indicate that the revision is of poor quality.

For the purpose of building a reputation system, we want a measure that provides some insight into the community perception of the quality of the edit. Having defined the notion of text survival, a very simple quality measure could be "what fraction of the text added in a revision survives ten revisions later?" We discuss some variations of these quality measures in Chapter 5, but present one novel quality measure here.

In trying to understand how text contributions evolve, we decided to limit our exploration to what happens to the text over the following ten revisions. Many contributions follow the simplest model: they are either removed completely right away (a revert), or they are perfectly preserved for the following ten revisions. Some contributions, however, are only partially preserved, and might even be partially restored as part of their evolution. Figure 4.1 gives a pictorial representation of how some text introduced at revision v_k might evolve over the next seven revisions; in this example, the figure shows that some text was restored in revision v_j. We say that author A_j *judges* the work of author A_k by deciding how much text to preserve, delete, or restore. If

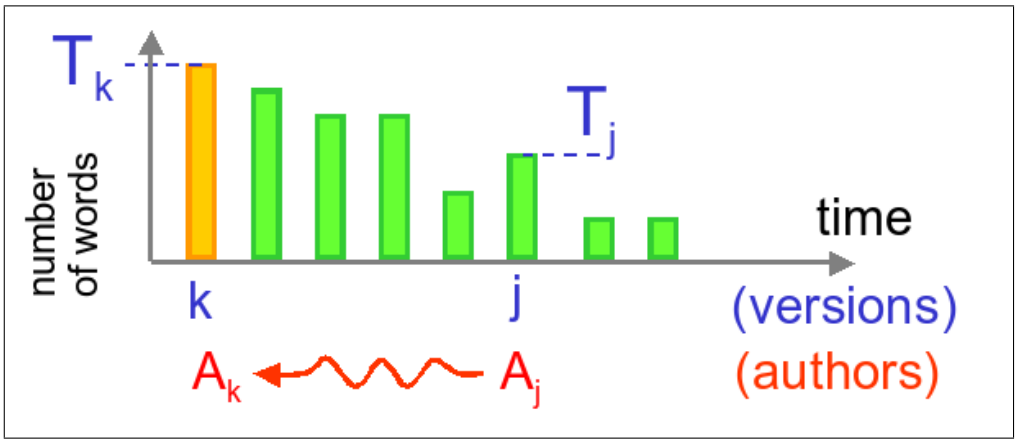


Figure 4.1: A graphical depiction of how a text contribution survives through future revisions. An author, A_k, adds $T_k = TSurv(k, k)$ words in revision v_k. In subsequent revisions, some of those words are deleted and partially restored. We say that a later author, A_j, implicitly *judges* author A_k by choosing how many of A_k's words to keep or delete or restore; $T_j = TSurv(k, j)$ is the number of words that were introduced in v_k still present or *live*, in v_j.

author A_j works on a different part of the article, she is still implicitly deciding that the current revision of A_k's work is okay.[8]

Measuring the fraction of text introduced in revision v_k that survives to revision v_j (i.e., computing $TSurv(k, j)/TSurv(k, k)$) gives useful information, but what if the author of v_j happens to be a vandal that blanks the page? One idea would be to average the text survival over the next several revisions (an idea explored in Chapter 5), but we were struck by the observation that after some initial churning, text seems to stabilize and then only slowly change as time progresses. We propose that one way to model this evolution of text over time is as a geometric sequence with a common ratio between zero and one; see Figure 4.2 for how such a sequence could approximate the text survival over several revisions. The intuition behind a geometric model is that if an edit is "bad," then most of the text will be removed right away. As time passes, the size

[8]Some better model of user attention would be useful for tempering the amount of judgement we infer from A_j when they are focused elsewhere in the article.

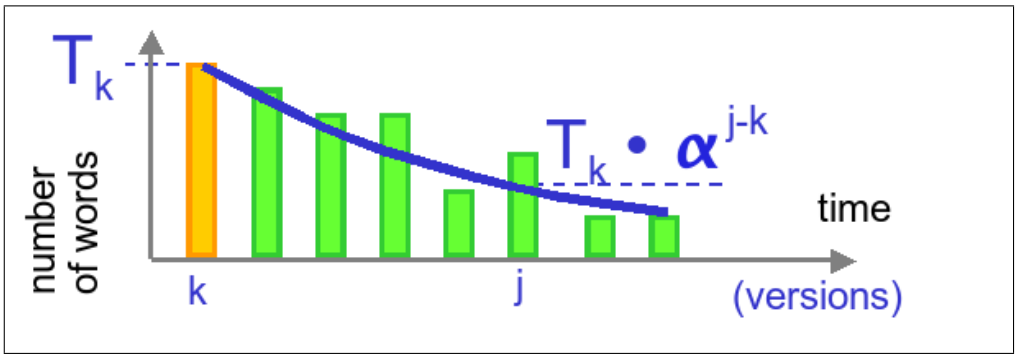


Figure 4.2: To calculate the *text longevity* of the contribution of T_k words, we model the text survival as a geometric curve and compute a single number that describes how the text evolves over several revisions. Value α is the solution to the function in Eq. 4.2 based on the geometric series.

of the edits decreases and the text tends to stabilize into a form that people can agree on until it eventually no longer changes.

To measure the overall quality of a single text contribution made in revision $v_k \in \mathbb{V}[a]$, we need to compute the common ratio that best describes the sequence of n text survival values, $TSurv(k, j)$, after v_k, where $k < j \leq k + n$. Let us call this common ratio the *text decay quality*, $q^n_{tdecay}(k)$. To compute $q^n_{tdecay}(k)$, we want to solve the following equation:

$$\begin{aligned}
\sum_{i=0}^{n} TSurv(k, k+i) &= TSurv(k,k) + q^n_{tdecay}(k) \cdot TSurv(k,k) \\
&\quad + (q^n_{tdecay}(k))^2 \cdot TSurv(k,k) + \ldots \\
&\quad + (q^n_{tdecay}(k))^n \cdot TSurv(k,k) \\
&= TSurv(k,k) \cdot \sum_{i=0}^{n} (q^n_{tdecay}(k))^i \\
&= TSurv(k,k) \cdot \frac{1 - (q^n_{tdecay}(k))^{n+1}}{1 - q^n_{tdecay}(k)}
\end{aligned}$$

To solve this for $q^n_{tdecay}(k)$, we can use Newton's method to solve for the zero of the related function:

$$f(\alpha) = -(1-\alpha^{n+1}) \cdot TSurv(k,k) + (1-\alpha) \cdot \sum_{i=0}^{n} TSurv(k,k+i) \tag{4.2}$$

where $\alpha = q^n_{tdecay}(k)$. Newton's method involves making repeated estimations of the form

$$\alpha_{j+1} = \alpha_j - \frac{f(\alpha_j)}{f'(\alpha_j)}$$

which we initiate with $\alpha_0 = 0$. For efficiency reasons, we limit the number of iterations taken to a small number (five in our live system) since we are only estimating the quality, and high precision is not very useful.

The beauty of this quality measure is that it varies between zero for text that is immediately deleted, and one, for text that is completely preserved. Values in between the two extremes reflect the fact that there was some debate among the community about what text to preserve in the article.

4.4 Edit Quality

The problem with measuring text contribution quality alone is that it measures only one kind of user behavior: inserting of text. Another important behavior of users is to rearrange text (possibly with minor edits) so that it improves the flow or readability of the text. In traditional publishing, this is a function provided by the editor — and both content creation and editing for grammar and style are valuable to the quality of the final product. Is there any notion equivalent to text survival for edits? In struggling to answer this question, we first had to measure the size of an "edit contribution," which

naturally led us to the *edit distance* [23, 58, 105, 20, 89] measure.

Edit Distances

Edit distance [23, 58] is typically used as a way to measure how many insertions, deletions, and replacements are needed to transform one string into another (the collection of these operations is called an *edit script*), and is usually defined as the sum of the number of those operations (that is, the size of the edit script). Other formulations exist for edit distance, and within the context of the WikiTrust project, we were already computing text differences as part of our author tracking algorithms, so we chose to define the distance between two revisions in terms of the edit script generated by our greedy text matching algorithm from Chapter 3. The elements making up the edit script generated by WikiTrust are:

- $\mathbf{Move}(i_1, i_2, k)$ – a block of text of length k words which matches between the source string and target string. The match starts at position i_1 in the source string, and starts at position i_2 in the target string.

- $\mathbf{Delete}(i, k)$ – the text starting at position i and extending for length k words was deleted from the source string.

- $\mathbf{Insert}(i, k)$ – the text starting at position i and extending for length k words was inserted into the target string.

If we let $E(m, n)$ be the edit script set of elements describing the transformation from the source string $\mathrm{Words}(v_m)$ to the target string $\mathrm{Words}(v_n)$, then we can define terms for the total amount of insertions and deletions by summing over all the matching

elements:

$$I_{tot}(m,n) = \sum_{\mathbf{Insert}(i,k)\in\mathbb{E}(m,n)} k$$
$$D_{tot}(m,n) = \sum_{\mathbf{Delete}(i,k)\in\mathbb{E}(m,n)} k$$

We must take more care in quantifying how much *movement* was involved in an edit; simply summing the size of each **Move** operation is not sufficient, because our edit script includes *all* matching text as these operations. What we would like is to only count those blocks of text that were actually rearranged. Let $l_m = |\mathrm{Words}(v_m)|$ and $l_n = |\mathrm{Words}(v_n)|$. Each time a block of text of length k exchanges position with a block of text of length k', we count this distance as $k \cdot k' / \max(l_m, l_n)$. Thus, a word that moves across k' other words contributes $k' / \max(l_m, l_n)$ to the distance; the contribution approaches 1 as the word is moved across the whole document. The total contribution from **Move** operations is then given by:

$$M_{tot}(m,n) = \sum_{\substack{\mathbf{Move}(i_1,i_2,k)\in\mathbb{E}(m,n) \\ \mathbf{Move}(i'_1,i'_2,k')\in\mathbb{E}(m,n) \\ i_1<i'_1 \wedge i_2>i'_2}} \frac{k \cdot k'}{\max(l_m, l_n)}$$

We then define the edit distance between revisions v_m and v_n as

$$d(m,n) = \max(I_{tot}(m,n), D_{tot}(m,n)) + M_{tot}(m,n) - \frac{1}{2}\min(I_{tot}(m,n), D_{tot}(m,n)) \tag{4.3}$$

The motivation for this formulation of edit distance is to try to account for replacements, which appear within the edit script as both insertion and deletions — but the position information required to match them is not preserved by the difference algorithm. By subtracting a correction term, we make the assumption that edits with both insertions and deletions make some replacements, which are being counted in both types of edit.

Edit Longevity

Given a method to measure the size of an edit contribution, the problem still remains of how to compute whether that contribution is preserved in future revisions. An idea we considered is summing the edit distances of sequential revisions, and comparing that against the edit distance between the first and last revisions. This has the problem that it roughly assumes that all contributions are completely preserved or completely reverted, but the idea of mapping out the edit distances of multiple revisions led to another idea that we favor.

Instead of trying to measure whether an edit is preserved, let us ask the question, "does this edit move us in the direction of the future of the article or does it move us away from the future?" Let us consider a physical analogy to see how this might work: suppose that grad school was as simple as a hike through the Forest of Research, with graduation on the far side of it (see Figure 4.3). The grad student starts off on a direct path towards graduation, but inevitably gets lost and wanders deep into the forest. At some point, the advisor gets involved and rescues the grad student, guiding him towards graduation. There are three distances that are involved in this scenario:

1. the work done by the grad student, from the start to the point where he is rescued,

2. the work done by the advisor, from the point of rescue until graduation, and

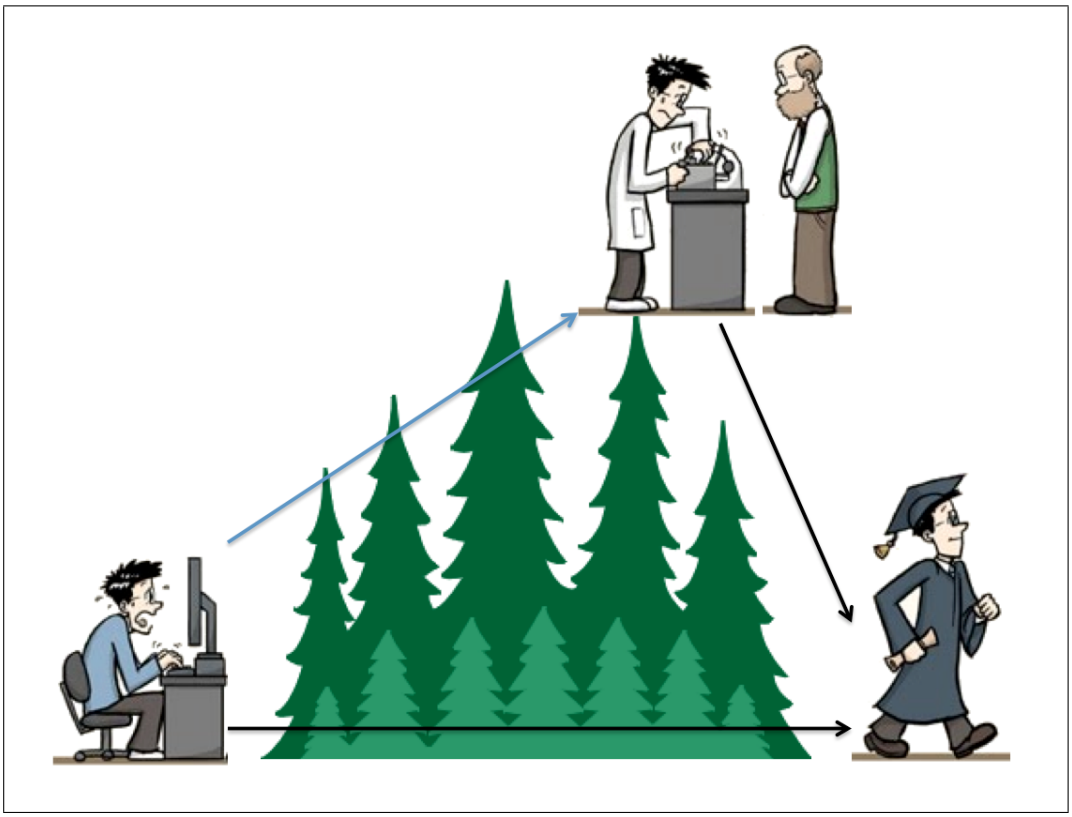
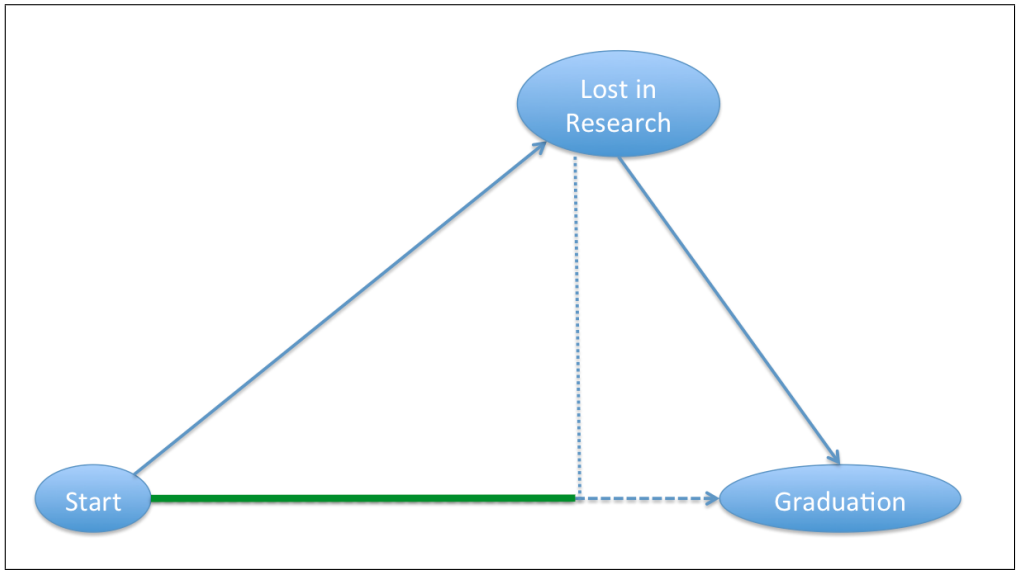


Figure 4.3: Grad school can be likened to hiking through a forest, where the grad student inevitably leaves the direct path and ends up far from his intended goal. When the advisor steps in with advice and guides the grad student to graduation, we ask the question "how much useful work did each contribute by their efforts?" (Character illustrations courtesy of [16].)

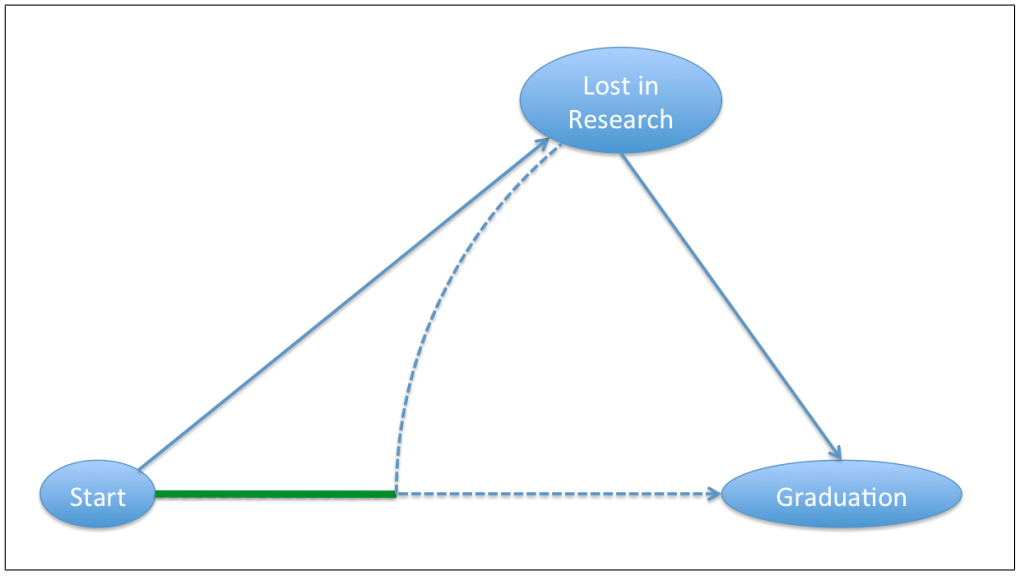
3. the direct path distance from start to graduation.

Our original question then becomes, "how much of the grad student's work contributes towards his reaching graduation?" From the grad student's perspective (Figure 4.4a), this is easy: a triangle is formed by the three distances, and to calculate the "useful work" one merely drops a normal from the point of rescue to the direct path; the remaining distance represents the contribution of the advisor. What if the student overshoots his target, so that the "forward progress" is more than the minimum necessary?

In the physical setting, the student has not reached the goal so we should reject



(a) The grad student measures his "forward progress" in the traditional way: by dropping a normal to the direct path.



(b) The advisor measures the useful effort of the grad student by subtracting his own total effort from the minimum effort that should have been necessary. From the advisor's perspective, his own effort is completely useful, and would have been unnecessary if the grad student had followed the path as directed.

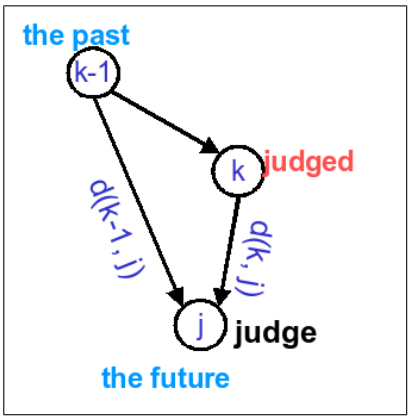
Figure 4.4: How should we calculate the useful effort that the grad student in Figure 4.3 contributed towards his own graduation? Here are two perspectives on how to measure that.

any computation that discounts the advisor's contribution. Reasoning that his time and expertise directly and efficiently contribute to guiding the student to the goal, the advisor proposes a different scheme (see Figure 4.4b) for computing the grad student's contribution: take the minimum work necessary to go from start to graduation and then subtract the amount of effort put in by the advisor. This has the effect of discounting the effort of the grad student, rather than the advisor, and has the benefit of a natural interpretation of when a contribution has "forward" (instead of "backward") progress.

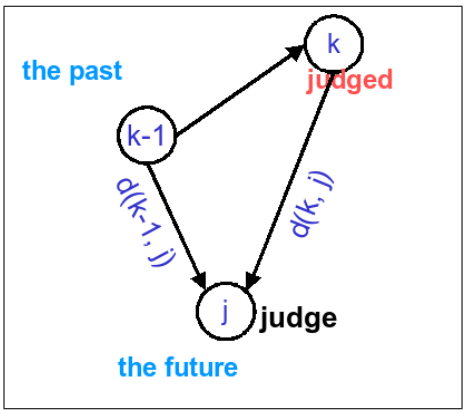
Applying this metaphor back to the Wikipedia, we must take into consideration three revisions: the one being evaluated, one in its past, and one in its future. We say that the edit being evaluated was useful if it brings the article closer to how it will look in the future. Of course, an author can "overshoot the future" by adding a mixture of content, some of which is kept and some of which is rejected; the efforts of the author are discounted by the amount of work that subsequent authors must do to transform article into its final (that is, some future) state, so that even a situation of overshooting the future has a reasonable interpretation.

Figure 4.5 visually represents two cases in evaluating revision v_k, using v_{k-1} and v_j (where $j > k$) as guide posts for the general path that the evolution of the article is taking. If a contribution moves us in the general direction of the future but has some extraneous text that is deleted (for example), we get the case shown in Figure 4.5a: the distance $d(k, j)$ is smaller than the distance $d(k-1, j)$. If the edits of v_k do not contribute at all to the future of the article, then we have the case shown in Figure 4.5b: the distance $d(k, j)$ is larger than the distance $d(k-1, j)$.

So now we have a very simple analysis of comparing $d(k, j)$ with $d(k-1, j)$ to tell us whether the quality of revision v_k is *good* or *bad* (at least with respect to v_{k-1} and v_j). We would like to have more gradation in a quality measure than just the two extremes;



(a) Graphical representation of a good edit contribution.



(b) Graphical representation of a bad edit contribution.

Figure 4.5: To measure the quality of version v_k, we also look at the previous version v_{k-1} and some future version v_j. The three versions form a triangle, using edit distance [23, 58] to define the separation between each other. Intuitively, we know that when v_k is good, the distance to the future, $d(k, j)$, will be shorter than if v_k is bad. (When v_k is bad, more editing is required to bring it back to a better version, plus the editing to bring it to the future.)

we would like to know *how* good or bad an edit is. To answer this question, we thought that a good measure would be to compare the total work done by $\mathrm{RevAuthor}(v_k)$, with the amount of progress made towards the future. That is, if an author writes a 600 word essay about a topic, but only 20 words are actually kept in the article, then the original 600 word contribution must have been pretty bad. Instead, if 300 words remain in the article (half of them), then the contribution could be said to be so-so.

We propose that a good way to measure this judgement of the author of v_k by the author of v_j, which we call *edit longevity*, is:

$$ELong(k, k-1, j) = \frac{d(k-1, j) - d(k, j)}{d(k-1, k)}, \tag{4.4}$$

It compares the "useful progress" $d(k-1, j) - d(k, j)$ with the "total work" $d(k-1, k)$. Figure 4.6 presents the graphical interpretation of the quantities involved. For reverted

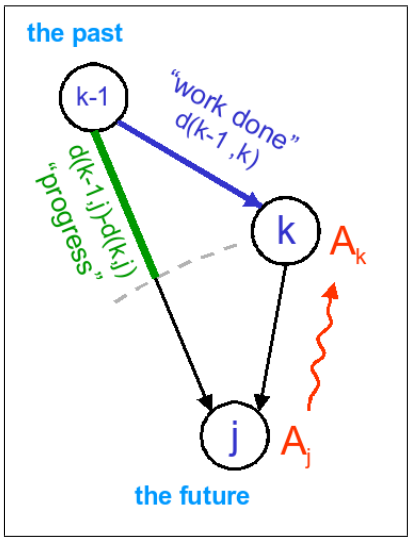


Figure 4.6: Quality is measured by calculating how much *progress* is made towards the future version of the article, and dividing that by the amount of *work done* during the edit.

edits, the ratio $ELong(k, k-1, j)$ is -1, since all the work goes into *increasing* the distance between v_k and v_j. For preserved edits, $ELong(k, j)$ is close to 1.

Note that here and in our original work on this topic [2], the reference revision from the past is always the revision immediately before the revision being evaluated. In practice, the revision immediately before might be vandalized, so that the path from v_{k-1} to v_j does not actually represent the overall trajectory of edits for the article. Another work dealing with the robustness of reputation [17] addresses this shortfall. Also, it is sometimes the case that a revision is created with no changes (or with changes only in whitespace), so that we have $d(k-1, k) = 0$; in this case, we ignore the triangle and do not compute a longevity score for revision v_k.

The Reverse Triangle Inequality

The insight leading to Definition 4.4 is that the edit distance measure induces a topological space that relates every revision to every other revision. Using that model for

our intuition, we can describe the evolution of an article from its creation to the most recent revision as a trajectory in the space. This is very much like a drunken walk along a road from one place to another; there is a clear start and a clear end, but an inebriated pedestrian won't follow a straight line. Using the straight line between start and end as a reference, we can measure the pedestrian's forward progress, as well as how much wasted effort he puts into moving sideways along the road. Figure 4.6 is applying that analogy in our edit distance based topology, and shows one way to measure the forward progress.[9]

Continuing with the physical intuition of how revisions can be related to each other, the reverse triangle inequality [122] allows us to put bounds on the value of Definition 4.4. The reverse triangle inequality states that any side of a triangle is greater than the absolute value of the difference of the other two sides:

$$|d(k-1,j) - d(k,j)| \leq d(k-1,k) \tag{4.5}$$

The two sides of this equation are exactly the two terms of our fraction, so that we can conclude that edit longevity varies from -1 to $+1$.

There is just one problem with this analysis: it depends on the triangle inequality holding. Unfortunately, this is not actually the case; we adjust for this by limiting the result to the range -1 to $+1$, with any value beyond that range being capped to the closest value within the range. Section 4.6 contains further discussion around this aspect of edit distance and our edit longevity definition.

[9]In trying to measure "the useful work done" there is the temptation to use the projection of $d(k-1,k)$ onto $d(k-1,j)$ by computing the angles involved. This isn't correct because of the possibility of *overshooting the future*. For example, an author might contribute completely good text but include an extra paragraph that is removed in the next revision; in this case, the triangle formed collapses to a straight line and we have $d(k-1,k) > d(k-1,j)$.

Edit Longevity Quality

Our strategy for turning edit longevity into a quality measure is to compute an average of edit longevity from several different perspectives. Edit longevity requires two landmarks to base a judgement on: some revision in the past of the version being judged, and some revision in its the future. For simplicity, we choose the revision immediately previous to the one being evaluated as the landmark in the past. That is, if we are evaluating v_k, then we choose v_{k-1} as one of the reference points. Our motivation for choosing only a single previous revision as a reference point is that past revisions cannot be making any implicit statements about a revision that has not happened yet.[10]

Our central idea for extracting community sentiment from the revision history of an article is to examine what happens to an author's contribution as the article continues to evolve. We say that later versions of the article act as *judges* of the contribution, because each later author implicitly makes the decision to delete or revise existing text in the article. For text quality we used the ten following revisions as judges, but this allows the possibility for the author being judged to act as his own judge. We address this issue in the case of turning edit longevity into a quality by defining a map

$$J : \mathbb{A} \times \mathbb{V} \times \mathbb{Z} \to 2^{\mathbb{V}}$$

which, for an article $a \in \mathbb{A}$, returns the set of up-to n versions that act as judges to version $v_k \in \mathbb{V}[a]$, with the proviso that the author of v_k does not act as his own judge:

$$J_a^n(k) = \{j : j \in \mathbb{Z} \,.\, 0 < j - k \leq n \wedge \mathrm{RevAuthor}(v_k) \neq \mathrm{RevAuthor}(v_j)\} \quad (4.6)$$

[10] In hindsight, the previous revision might have been blanked or otherwise vandalized, so that multiple previous revisions should actually be considered. See [17] for alternative methods of choosing landmark revisions for estimating the evolution of an article.

With our reference points for the past and future selected, we can define the average edit longevity as follows:

$$q_{elong}^{a,n}(k) = \frac{1}{|J_a^n(k)|} \cdot \sum_{i \in J_a^n(k)} ELong(k, k-1, i) \tag{4.7}$$

4.5 Evaluation

A large challenge in creating quality measures is the problem of evaluating the performance of the measure. "Quality" is an imprecise notion by itself, because it necessarily must be evaluated with respect to some attribute. For example, within the Wikipedia we might evaluate the quality of a contribution along any of these dimensions:

- grammar
- diction
- neutral point of view
- factual correctness

Our text and edit longevity measures try to go one step further into the fuzzy world of human evaluations by using later edits as a basis for inferring sentiment about earlier edits. Is this a valid inference to make? How can such a question be answered?

The surest way to measure sentiment would be to interview users as they are making edits to the Wikipedia and documenting their thought processes as they read an article and make the decision to edit an article — but this would require an enormous effort to collect enough data for performance evaluation. We propose that we can grossly measure the sentiment of the community by recognizing that there is a generally agreed

upon standard of articles being of "encyclopedic quality" which allows people to recognize vandalism when they see it. Accepting that premise suggests that we can use the PAN-WVC-10 corpus [78] as a manually annotated data set for such an evaluation.

The PAN-WVC-10 corpus was used to compare the performance of solutions for the First International Competition on Wikipedia Vandalism Detection (PAN-WVD 2010) [80]; We use it in a similar way here to compare how well our quality measures are able to predict vandalism within the corpus, but with an important distinction: we use information "from the future" to calculate our quality values for the annotated revisions. Standard vandalism detection tools make their determination immediately as the edit is made, so that any vandalism can be quickly repaired by other users. The necessity for a quick classification precludes waiting for future edits or rating to corroborate the edit being judged; we term this variation of the problem *immediate vandalism detection*. By construction, our two longevity metrics use later edits to measure the quality of the revision being judged; we call this the *historical vandalism detection* problem. Historical vandalism detection has its own set of important applications, such as selecting high quality revisions for DVD compilations or for presentation to school children.

The PAN-WVC-10 corpus contains 32,439 edits, where each revision was manually reviewed by at least three annotators to assign a label of either "regular" or "vandalism." We used the dump of the English Wikipedia from January 30, 2010 to extract the text of each annotated revision, along with the revision before and the ten filtered revisions following so that we could compute our text longevity and edit longevity measures for each annotated edit.

We used the straight-forward transformation to convert each quality score from its normal range into the range $[0, 1]$, to be interpreted as a probability that the named revi-

sion was the result of vandalism. As in the PAN-WVD 2010 competition [80], we use the `perf`[11] package to evaluate the performance of our quality measures by computing the areas under the receiver operating characteristic curve[12], and the precision-recall curve.

Difference Algorithms

The formula for edit distance we defined in Section 4.4 is calculated from the operations within the edit script describing the transformation from the source revision to the target revision. This edit script is highly dependent on the algorithm used to compute the difference between the revisions. To provide a more complete picture of how the choice of difference algorithms affects the performance of the quality measures, we present an evaluation of some variations of the algorithms. Our evaluation for this chapter was implemented in OCaml for performance reasons; the OCaml source representing these algorithms appears in Appendix E.

The basis for the WikiTrust method of text differencing is a greedy algorithm inspired by the work of Tichy [105]. As described in Chapter 3, our goal was to create an edit script that more closely followed a human understanding of the text transformation, rather than trying to optimize for shortest edit script. This led us to change the greedy step to be *globally greedy*; that is, we always prefer the longest unmatched sequence of words from anywhere in the source and target strings. This differs from the greedy step of the Tichy method, where the target string is constructed from left to right by selecting the longest match for the next unmatched position in the target. To judge the efficacy of this choice, we implement both methods as part of our evaluation.

[11] `http://osmot.cs.cornell.edu/kddcup/software.html`

[12] `http://en.wikipedia.org/wiki/Receiver_operating_characteristic`

min words	minimum length of match from source to target.
max matches	maximum number of matching substring prefixes.
longest match	only place longest match onto priority queue.
prev matches	use matches from previous position to determine longer match.
header/trailer	match beginning and end of article first.

Table 4.1: Summary of WikiTrust differencing optimizations. See Section 3.4 for a detailed explanation of each optimization.

For the Tichy method, we implement the optimizations suggested by Obst [72] and Reichenberger [85].

In Section 3.4, we propose several different optimizations to improve the running time of the WikiTrust method and we include a few combinations of these optimizations in our evaluation; see Table 4.1 for a summary of the optimizations, and Table 4.2 for a matrix of how we labeled the combinations of optimizations in our evaluation. Note that we also include for comparison **diff6** and **diff7**, which are the exact implementations of the WikiTrust differencing algorithm used in our production service. These two implementations are based on the same ideas explained in Chapter 3, but use code originally written for [2] with extra optimizations not described here.[13]

Match Quality Formulas

The match quality formula is the key to our greedy algorithm choosing matches which meet the desired properties outlined in Chapter 3. We experimented with the nine definitions enumerated below, which take as parameters the length of the match (parameter k), the matching positions in the source and target revisions (parameters i_1 and i_2), and

[13]Both functions are in file `chdiff.ml` and can be examined from our Github repository at `http://www.github.com/collaborativetrust/WikiTrust`. Algorithm **diff6** corresponds to function `edit_diff_core`, and **diff7** corresponds to function `edit_diff_live`.

Algorithm	min words	max matches	header/trailer	longest match	prev matches	Description
diff1	3	50				Tichy method
diff2	3	50	x			Tichy method
diff3	3	50	x	x		WikiTrust method
diff4	3	50		x	x	WikiTrust method
diff5	3	50	x	x	x	WikiTrust method
diff6	3	50		x	x	WikiTrust method, production
diff7	3	50	x	x	x	WikiTrust method, production
diff8	3	50		x		WikiTrust method
diff9	3	50	x			WikiTrust method

Table 4.2: Listing of optimizations used by each difference algorithm. See Table 4.1 for a one-line summary of each optimization. Note that **diff9** results are not presented in this chapter because of its prohibitive running time; some results at the revision level are given in Table 3.1.

the total length of each revision (parameters l_1 and l_2); a "chunk index" (parameter c[14]) is also passed to the match quality function, but is only used in the computation of text longevity. The match qualities computed by our system are tuples, with lexicographically smaller tuples considered to be of higher quality.

The initial set of match quality functions we present form the baseline functions of what could be used in conjunction with the standard greedy algorithms for text differencing. They use the length of the match and the chunk index in various combinations. In these, and later match quality functions we define, we mark with a dagger (†) those functions which are not compatible with the assumption that longest matches are

[14]The *chunk index* refers to which chunk a block of text comes from. Chunk 0 refers to the full text of the older revision being compared. Chunk 1 refers to text from the revision previous to that which was deleted in the edit that resulted in the revision of chunk 0. Chunk 2 refers to deleted text from the revision previous to the revision of chunk 1, and so on. This is an optimization described in Section 3.4 for more efficient calculation of text authorship. In computing the edit distance, differences are only computed between two revisions, so we always have $c = 0$.

preferred, used in our **longest match** optimization to the difference algorithms. The implementation of each match quality function can be examined in Appendix E; the function names are of the form `quality_#`.

mq1 This version uses a match quality tuple of $(-k, c, 0)$.

mq2 This version uses a match quality tuple of $(-k, -c, 0)$, changing the sign of the chunk index to instead prefer matches with older revisions. Since the chunks older than chunk 0 are only fragments of deleted text, potentially longer matches in more recent text will be missed; for that reason, we expect this match qualiy function to perform worse than **mq1**.

mq3 This version uses a match quality tuple of $(c, -k, 0)$, making the matching revision the dominant factor. †

mq4 This version uses a match quality tuple of $(-c, -k, 0)$, reversing the preference for matches in the most recent revision. We expect this match quality function to perform worse than **mq3**. †

The next set of match quality functions we present are based on the match quality described in the original WikiTrust publication [2].

mq5 Define

$$q := \frac{k}{\min l_1, k_2} - 0.3 \cdot \left| \frac{i_1 + \dfrac{k}{2}}{l_1} - \frac{i_2 + \dfrac{k}{2}}{l_2} \right|$$

The match quality tuple generated by **mq5** is $(0, -c, -q)$. † This is exactly the match quality used in [2].

mq6 As with **mq5**, define

$$q := \frac{k}{\min l_1, l_2} - 0.3 \cdot \left| \frac{i_1 + \frac{k}{2}}{l_1} - \frac{i_2 + \frac{k}{2}}{l_2} \right|$$

To make **mq5** compatible with the **longest match** optimization to the difference algorithm, we introduce the match length as the primary discriminant: the match quality tuple for **mq6** is $(-k, -c, -q)$.

mq7 This is a modification of **mq6** to change the priority of the chunk index. Define

$$q := \frac{k}{\min l_1, k_2} - 0.3 \cdot \left| \frac{i_1 + \frac{k}{2}}{l_1} - \frac{i_2 + \frac{k}{2}}{l_2} \right|$$

The match quality tuple is $(-k, c, -q)$.

The difficulty in working with the original WikiTrust match quality functions, **mq5**-**mq7**, is that the definition of q is doing too much. It tries to balance length of the match as a fraction of the document length, against the relative positioning of the matches. To get this balance right requires much experimentation across many documents, and even then might require revision as the dynamics of group collaboration in a document change over time. As part of the **longest match** optimization for the difference algorithm, it became necessary to separate the concern for match length from that of relative positioning, resulting in our current scheme of tuples to represent the match quality. This final set of match quality functions tests two variations of that idea:

mq8 This is the quality function used in the live production WikiTrust system. Define

$$q' := \left| \frac{i_1 + \frac{k}{2}}{l_1} - \frac{i_2 + \frac{k}{2}}{l_2} \right|$$

This computes the midpoint of each end of the match, and then compares their relative positions within the source and target revisions. The closer to the same relative position each end of the match is, the closer to zero q' gets. The final match quality tuple used is $(-k, -c, q')$.

mq9 This modification of **mq8** changes the priority of the chunk index to test the effect on text longevity.

$$q' := \left| \frac{i_1 + \frac{k}{2}}{l_1} - \frac{i_2 + \frac{k}{2}}{l_2} \right|$$

The final match quality tuple returned is $(-k, c, q')$.

Edit Distance Formulas

The edit longevity measure uses edit distance as a proxy for the amount of work that an author puts into an edit. We tested the following definitions of edit distance to understand how this choice impacts the quality of the results.

ed1 This edit distance computes the sum of the lengths of insertions and deletions, that is, the amount of text which is either added or deleted from one revisions to the next:

$$I_{tot} + D_{tot}$$

ed2 Traditionally, edit distance functions incorporate all the elements of an edit script. We define this edit distance to compute the sum of the lengths of insertions, deletions, and move operations.

$$I_{tot} + D_{tot} + M_{sum}$$

where

$$M_{sum}(m, n) = \sum_{\mathbf{Move}(i,j,k) \in \mathbb{E}(m,n)} k$$

Note that any text[15] which appears in both revisions which are being compared will be counted as a move operation, even if it has not relocated within the document. For example, if there is no difference between two revisions of an article, the edit script will consist of a single **Move** operation that is the size of the revisions. Thus, we expect that this function will perform poorly because it will be dominated by the size of the article.

ed3 The WikiTrust difference algorithms do not keep track of replacements in the text; instead, a replacement appears in the edit script as both a deletion and corresponding insertion at the same point in the text. We modify **ed1** to apply a correction factor which assumes that some part of the work is always a replacement:

$$I_{tot} + D_{tot} - \frac{\min(I_{tot}, D_{tot})}{2}$$

ed4 This is the edit distance described in Section 4.4, first proposed in the original

[15] Technically, this should be "most text," because of the **min words** optimization.

WikiTrust paper [2].

$$\max(I_{tot}, D_{tot}) - \frac{\min(I_{tot}, D_{tot})}{2} + M_{tot}$$

ed5 This is the edit distance computation used in the live production system of WikiTrust. It models the matches between the two strings as a graph and uses connected components to try to ascertain when an insertion and deletion pair are actually replacements. We include the evaluation of this edit distance formula for comparison purposes, but detailing the specifics of this algorithm falls outside the scope of this work. Interested readers can examine the source code, available at Github,[16].

Results

A complication in our evaluation is our restricted setting of *filtered* revisions, where sequential revisions by the same author are filtered out to leave only the last revision in the sequence. This would limit us in evaluating the performance of our quality measures, so we modified the system in the following way: we do not filter the specific revisions annotated in the PAN-WVC-10 corpus, or the immediately preceding revision (even when they have the same author), but we do filter revisions *after* the annotated revision in the usual way. Even with this loosening of the revision filtering, several revisions are still not evaluated for quality; the two primary reasons for no evaluation are a lack of subsequent edits to base the evaluation on, or the revision was not substantially different from the previous revision.[17]

[16] `http://www.github.com/collaborativetrust/WikiTrust`

[17] That is, when $d(k-1, k) = 0$. This typically happens when there are only whitespace changes from one revision to the next, but some of the proposed edit distance measures don't consider the entirety

Edit Longevity

Diff	MatchQuality	PR-AUC
diff2	mq6789	47.279%
diff2	mq1234	47.258%
diff3	mq5	47.231%
diff5	mq5	47.231%
diff8	mq5	47.198%
diff4	mq5	47.198%
diff3	mq6789	47.192%
diff5	mq6789	47.192%
diff2	mq5	47.171%
diff3	mq1234	47.152%
diff8	mq6789	47.140%
diff4	mq6789	47.140%
diff4	mq1234	47.124%
diff8	mq1234	47.111%
diff5	mq1234	47.095%
diff6	mq1234	47.041%
diff1	mq6789	47.037%
diff1	mq5	47.035%
diff1	mq1234	46.990%
diff6	mq6789	46.943%
diff7	mq1234	46.583%
diff6	mq5	46.552%
diff7	mq6789	46.539%
diff7	mq5	46.100%

Table 4.3: Performance of difference algorithms for edit distance **ed5**. Where multiple match quality functions resulted in the same performance, they have been grouped together; for example, **mq6789** represents **mq6**, **mq7**, **mq8**, and **mq9**.

The chief measure of performance that we consider is the area under the precision-recall curve (PR-AUC); we summarize the data here, with the full data available in Appendix F. We chose this as the primary measure because it gives better discrimination between predictive models for the PAN 2010 corpus, as explained in [80]. In the case

of the edit script and will have a higher incidence where the distance from the previous revision is zero.

Diff	MatchQuality	PR-AUC
diff4	mq5	39.279%
diff8	mq5	39.279%
diff4	mq6789	39.251%
diff8	mq6789	39.251%
diff4	mq1234	39.187%
diff8	mq1234	39.162%
diff6	mq1234	39.129%
diff5	mq5	39.109%
diff3	mq5	39.109%
diff3	mq6789	39.091%
diff5	mq6789	39.091%
diff6	mq6789	39.059%
diff5	mq1234	39.022%
diff3	mq1234	39.010%
diff1	mq6789	38.790%
diff1	mq5	38.783%
diff1	mq1234	38.728%
diff2	mq1234	38.670%
diff2	mq6789	38.664%
diff2	mq5	38.659%
diff6	mq5	38.559%
diff7	mq6789	37.785%
diff7	mq1234	37.652%
diff7	mq5	37.289%

Table 4.4: Performance of difference algorithms for edit distance **ed4**. Where multiple match quality functions resulted in the same performance, they have been grouped together; for example, **mq6789** represents **mq6**, **mq7**, **mq8**, and **mq9**.

Diff	MatchQuality	PR-AUC
diff1	mq6789	44.464%
diff1	mq5	44.456%
diff1	mq1234	44.450%
diff4	mq5	44.240%
diff8	mq5	44.240%
diff2	mq5	44.155%
diff4	mq6789	44.141%
diff8	mq6789	44.141%
diff4	mq1234	44.122%
diff6	mq1234	44.106%
diff8	mq1234	44.099%
diff5	mq5	44.067%
diff3	mq5	44.067%
diff7	mq6789	44.052%
diff6	mq6789	44.042%
diff2	mq1234	44.022%
diff2	mq6789	44.020%
diff7	mq1234	43.988%
diff5	mq12346789	43.970%
diff3	mq6789	43.970%
diff3	mq1234	43.935%
diff7	mq5	43.889%
diff6	mq5	43.774%

Table 4.5: Performance of difference algorithms for edit distance **ed3**. Where multiple match quality functions resulted in the same performance, they have been grouped together; for example, **mq6789** represents **mq6**, **mq7**, **mq8**, and **mq9**.

of our variations, PR-AUC is highly correlated with ROC-AUC, so the choice does not largely impact the ordering of the different algorithms. Our methodology for calculating PR-AUC and ROC-AUC is important to make clear: a standard evaluation (as was used in the PAN-WVD 2010 competition) requires predictions for all 32,439 edits in the PAN-WVC-10 corpus; we measure the performance for each combination of parameters only according to the number of revisions that a prediction is available for, which varies in a range from 27,500 to 28,500. As mentioned previously, we will not

Diff	MatchQuality	PR-AUC
diff7	mq6789	30.210%
diff7	mq1234	30.193%
diff7	mq5	30.146%
diff5	mq5	30.034%
diff3	mq5	30.034%
diff3	mq6789	30.020%
diff5	mq6789	30.020%
diff6	mq5	30.015%
diff3	mq1234	30.010%
diff5	mq1234	30.008%
diff2	mq6789	29.986%
diff6	mq6789	29.983%
diff6	mq1234	29.963%
diff2	mq12345	29.962%
diff4	mq1234	29.897%
diff8	mq5	29.895%
diff4	mq5	29.895%
diff8	mq6789	29.893%
diff4	mq6789	29.893%
diff8	mq1234	29.892%
diff1	mq6789	29.847%
diff1	mq5	29.836%
diff1	mq1234	29.803%

Table 4.6: Performance of difference algorithms for edit distance **ed2**. Where multiple match quality functions resulted in the same performance, they have been grouped together; for example, **mq6789** represents **mq6**, **mq7**, **mq8**, and **mq9**.

calculate a prediction when the edit distance from the previous revision is zero, or when there are no suitable judging revisions after the edit being judged. By computing the performance according to the number of revisions for which a prediction is available there is a slight bias towards parameter combinations which make fewer predictions, but this allows comparison of each combination in their best possible light without the distortion of a default guess when there is not enough data.

The most striking thing about the results presented in Appendix F is the clustering

Diff	MatchQuality	PR-AUC
diff7	mq6789	43.033%
diff7	mq1234	43.015%
diff7	mq5	43.002%
diff6	mq1234	42.891%
diff6	mq6789	42.843%
diff6	mq5	42.744%
diff2	mq5	42.665%
diff1	mq5	42.662%
diff5	mq5	42.617%
diff3	mq5	42.617%
diff5	mq1234	42.561%
diff3	mq1234	42.541%
diff5	mq6789	42.535%
diff3	mq6789	42.535%
diff1	mq1234	42.512%
diff1	mq6789	42.502%
diff2	mq6789	42.460%
diff2	mq1234	42.437%
diff4	mq5	42.415%
diff8	mq5	42.415%
diff4	mq1234	42.355%
diff8	mq6789	42.344%
diff4	mq6789	42.344%
diff8	mq1234	42.332%

Table 4.7: Performance of difference algorithms for edit distance **ed1**. Where multiple match quality functions resulted in the same performance, they have been grouped together; for example, **mq6789** represents **mq6**, **mq7**, **mq8**, and **mq9**.

by edit distance (and to a much lesser extent, of difference algorithm) that occurs. Clearly revealed is that use of **ed5** leads to the most superior predictions, while **ed2** makes the worst predictions (and **ed4**, the second worst). These results align with the intuition that the choice of edit distance function measures of the amount of *effort* done by an author in making an edit; the more closely that choice matches the human intuition, the better the ability to compute how much effort is *useful*. In particular, **ed2** was expected to perform poorly because its inclusion of all **Mov** operations leads to a value that is dominated by the size of the revisions, and **ed5** tries the hardest to estimate the amount of effort that goes into rearranging blocks of text.

In Tables 4.3 through 4.7, we present the predictive performance of the algorithms, controlling for edit distance. We find that match qualities **mq1**-**mq4** and **mq6**-**mq9** are always grouped together (shown in the tables as **mq1234** and **mq6789**). That is because these match qualities only differ from each other in how they treat the chunk index, which is not used in the edit longevity computation.

Diff	Avg Run Time	Std Dev RT
diff2	95m	0.57m
diff1	95m	0.64m
diff5	159m	6.53m
diff3	209m	6.51m
diff7	211m	50.05m
diff4	241m	13.10m
diff8	334m	13.91m
diff6	389m	105.54m

Table 4.8: Average running time of difference algorithms.

The typical measure used to compare difference algorithm is the running time, which we also measured. We find in Appendix F that the running time is usually little affected by either the match quality or the edit distance calculations, so we summarize

the results in Table 4.8 by computing the average and standard deviation of running times. The running time measured is the time required to run the WikiTrust analysis on the reduced corpus that includes the PAN-WVC-10 revisions; this includes the time to compute both edit longevity and text longevity for each revision, but not the time to sort the resulting statistics and generate reputation scores for authors. (Text longevity uses a fixed difference algorithm and only varies the match quality function, so that it is essentially a constant overhead in the time.) The most striking feature of Table 4.8 is the unusually large standard deviations of **diff6** and **diff7**. These two algorithms turn out to have a bimodal distribution.

In terms of predictive performance of the difference algorithms themselves, there is no clear winner. The top two edit edit distance functions, **ed5** and **ed3**, work best with the variations of the Tichy method of differencing — but even considering only **ed5** (Table 4.3) and **ed3** (Table 4.5), the ranking between **diff1** and **diff2** is exchanged. Even more telling, **diff1** gives the worst performance in the entire experiment when combined with **ed2** (Table 4.6), implying that no single difference algorithm is the best choice for the WikiTrust application.

With respect to the different optimizations, Table 4.8 provides some insights. The most notable conclusion is that Reichenberger's variation of the Tichy difference algorithm is faster than the WikiTrust greedy algorithm by nearly a factor of two.[18] With hindsight, we expect this sort of result because while both algorithms inspect every possible match and then mark the actual match as used, the WikiTrust algorithm additionally re-traverses each match to verify it or search for *residual* matches.

In our original justification for the **longest match** optimization, we argued that it

[18]The live production version of our WikiTrust code includes another technique for speeding up the difference computation, so that the performance more closely rivals the runtime of the Reichenberger versions.

would save CPU time to not have to test all the partial matches. We attempted to test this idea by implementing a version of the WikiTrust algorithm without the **longest match** optimization, and found that the memory requirements were too great to execute on our machines. Although the optimization is clearly a useful one to make, we note that Table 4.8 shows that the Tichy algorithm with optimizations (**diff1** and **diff2**) still has a significant advantage in runtime performance, and very similar vandalism prediction performance compared to the WikiTrust variations.

The ordering of **diff3**, **diff4**, and **diff5** allows us to infer that the **header/trailer** optimization is more beneficial than the **prev matches** optimization. Incorporated into the Tichy algorithm, the **header/trailer** optimization makes little difference to the running time.

The caveat to the use of the **header/trailer** optimization is that it can lead to different edit scripts being computed to describe the transformation from one revision to the next. Our hypothesis was that the **header/trailer** optimization more naturally conforms to a human description of how to transform one revision into another, leading to improved predictions of vandalism; this is not born out by the data in Tables 4.3 through 4.7. There are three pairs of difference algorithms that are the same except for the use of the **header/trailer** optimization: **diff1/diff2**, **diff4/diff5**, and **diff6/diff7**. Examining the data, there is no clear advantage to including or not including the **header/trailer** optimization in terms of predictive ability, but it does make a significant improvement in the running time for the WikiTrust difference algorithms.

We proposed the **prev matches** optimization with the expectation that it would reduce the memory requirements of the algorithm without substantially changing the performance with respect to predictive ability. Comparing **diff5**, which implements **prev**

matches, with **diff3**, which does not, we find that the predictive ability is slightly different for match qualities **mq1** through **mq4**. We examined this difference more carefully and found that the priority queue implementation will sometimes return matches in a different (but equivalent with respect to the priority) order. Match quality **mq8** is defined in a way that eliminates this reordering possibility, and we see that **diff3** and **diff5** always perform the same in this case.

During our exploration of the differences between **diff5** and **diff3**, we also found that the **prev matches** optimization interacts with the **max matches** optimization. By ignoring overly common string prefixes, the "previous match" list is empty for the next starting position after the common string prefix. If there is a long match which spans several overly common prefixes, **prev matches** loses track that it is traversing a region which already had a longer match and needlessly adds smaller matches to the priority queue, creating extra work. The timing information presented in Table 4.8 reveals that **prev matches** still has the advantage, but there are perhaps more performance gains to be had here.

Text Longevity

For measuring the performance of text longevity, we fixed the text matching algorithm to be the one used in the live production system of WikiTrust, which is a variation of the difference algorithm **diff7**.[19] Also, the edit distance formula does not affect the text longevity calculation, so our only significant parameter is match quality; results are presented in Table 4.9. Each variation made predictions for 28,453 revisions.

The match quality function has only a very small influence on the performance,

[19]The source code implementing this functionality is available from Github, `http://www.github.com/collaborativetrust/WikiTrust`, and corresponds to the function `text_tracking` in file `chdiff.ml`.

mirroring the small influence it has on edit longevity. We believe this is because the possible edit scripts for each edit due to the different match quality functions have a high degree of overlap; that is, the different algorithms generally produce the same edit script except for very minor differences.

Match Quality	PR-AUC	ROC-AUC
mq5	29.312%	85.980%
mq4	29.298%	85.972%
mq3	29.271%	85.962%
mq2	29.241%	85.950%
mq1	29.236%	85.948%
mq8	29.235%	85.949%
mq6	29.235%	85.949%
mq9	29.218%	85.942%
mq7	29.218%	85.942%

Table 4.9: Comparison of text longevity performance using multiple match quality functions, sorted by PR-AUC.

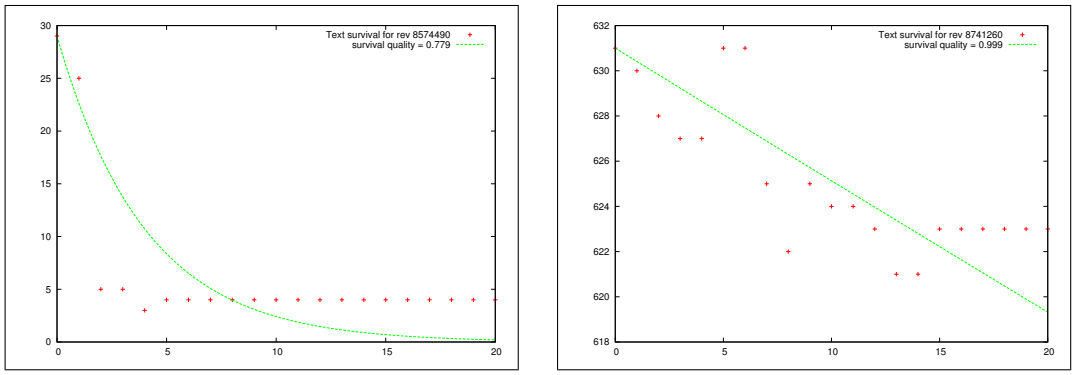
4.6 Additional Analysis

The results presented in the last section raised two additional questions. In comparison to edit longevity, the performance of $\approx 29.26\%$ PR-AUC by text longevity is quite poor; why does text longevity not do so well as edit longevity? Our second question is about the triangle inequality: how important is it that our edit distance function satisfy this property to be a good predictor of vandalism?

Edit Longevity Outperforms Text Longevity

As part of our investigation, we started looking at specific instances of text longevity values. In Figure 4.7, we see the text survival for two different contributions; both

do seem to have the general "exponential" shape that we previously described. Also computed in each figure is the text longevity measure based on the 20 revisions shown in each graph, but notice that the text longevity computed for Figure 4.7a doesn't exhibit the curve we expect. Instead of following the text survival, the curve goes below the level of text which survives each revision.



(a) The text survival graph for the text contributed early in the history of article George W. Bush.

(b) The text survival graph for the text initially contributed as part of the article Santa Cruz Beach Boardwalk.

Figure 4.7: The text survival quality of two different articles, computed based on 20 revisions. The majority of the editing happens in the revisions immediately after the initial edit, in these two cases.

The explanation for this discrepancy turns out to be a flaw in our thinking about the original model. While the text survival for contributions does seem to have an exponential look to it, exponentials do not approach some fixed non-zero value — they approach zero. In order to fit the curve to closely follow the points of text survival, the last value (in the case of the data shown in Figure 4.7a, the amount of text that survives after the 20^{th} revision) should be taken as the "zero reference point" which is subtracted from all the values. Applying our exponential curve fitting technique to these new values will give a much better approximation to the data. The problem with this better fit is that it changes the meaning of a score of zero; instead of meaning that the text was immediately deleted, a score of zero would mean that the text immediately

reached its final survival level. In other words, we would be measuring how quickly the text stabilizes, rather than how much agreement there was that the text belonged in the article.

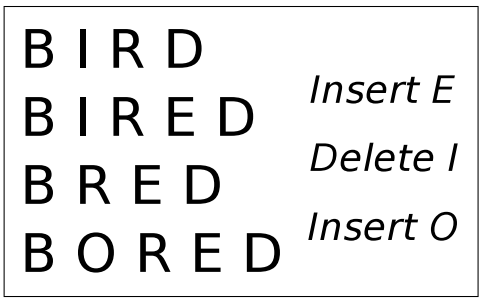
The Triangle Inequality

The intuition behind our formulation of edit longevity relies on the metaphor analogizing the *distance* between two revisions with the *work* or *effort* that an author puts into making the edit from one revision to the other; in particular, it is the triangle inequality (one of the metric properties of distance) that allows us to say that we can compute how much effort was *useful* in bringing the article closer to how it appears in a future revision.

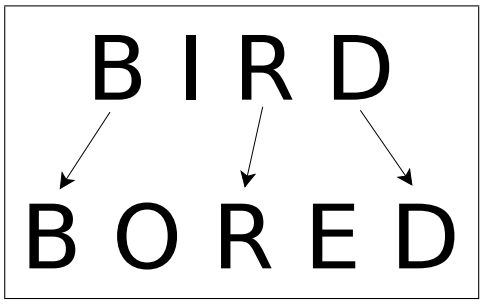
We have explored the use of several different definitions of the *edit distance* to represent this effort, but noted that the triangle inequality did not completely hold; see [89] for a summary of known conditions under which the triangle inequality holds for *listing*, *alignment*, or *trace* distances.

Our difference algorithm makes use of Tichy's block moves [105], which amounts to computing the trace that matches the source string to the target string. That matches are allowed between any parts of the two strings is equivalent to allowing transpositions as well as the usual insert and delete operations; the difference we compute does not support substitutions of one word with another. There is previous research on allowing transpositions [61, 109, 89] in computing edit distance; the goal of our work differs from this earlier work in that we prefer to select longest matches rather than minimizing the total edit distance computed.

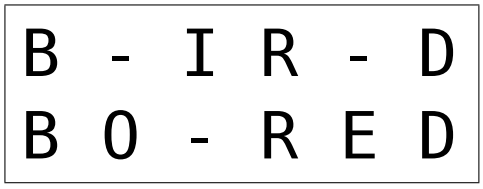
The various proposals for edit distance that we investigated are computations derived from the edit script we compute. Tichy's original counter-example shows that



(a) Listing distance is computed by finding the shortest edit script.



(b) Simple trace distance is computed by establishing a correspondence between the source and target strings, such that trace lines do not cross.



(c) Alignment distance is computed by finding an alignment which results in the minimum number of insertions and deletions.

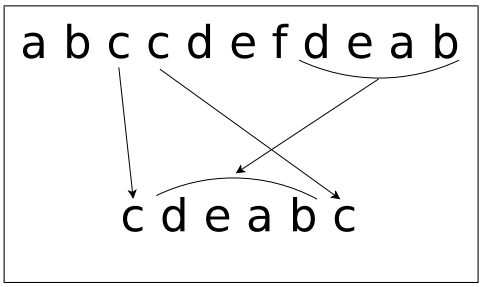
Figure 4.8: Examples of the three different methods typically used to compute edit distance. See [89] for an in-depth discussion on the distinctions between these methods.

globally greedy algorithms such as ours do not compute the minimum edit script (see Figure 4.9), making it unlikely that our proposed edit distance formulas guarantee the triangle inequality.

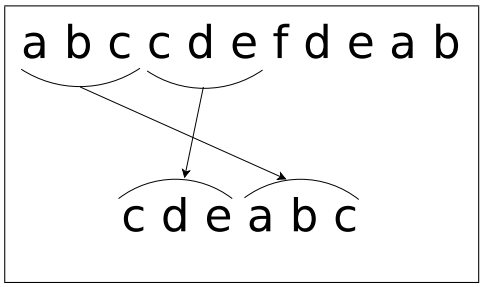
We present in Appendix F data on the frequency with which the triangle inequality holds for the various combinations of difference algorithms and edit distance formulae; only formula **ed2** never violates the triangle inequality. This is relatively easy to see by an enumeration of the cases, but it is instructive to examine **ed1** first.

To show that **ed1** does not satisfy the triangle inequality, we note that the definition

$$I_{tot} + D_{tot}$$



(a) The matching obtained using the WikiTrust method of determining all matches and selecting the longest matches first.



(b) The matching obtained using the Tichy method of scanning the target string from left-to-right and selecting the longest match found in the source string.

Figure 4.9: An example of how the matching between a source string and target string can differ between WikiTrust's greedy preference for the longest match anywhere between the two strings, and Tichy's processing of the target string from left-to-right and selecting the longest match for the given starting position in the target string. The WikiTrust algorithm describes the match as three operations, while the Tichy algorithm is able to match with two operations. This example is based on Tichy's original example demonstrating that a globally greedy algorithm does not result in the minimum number of operations [105].

is amenable to the *weighted operation* notation used throughout [89]. In this notation, an insertion of word x is said to contribute weight $w(\phi, x)$ to the final edit distance, and deletion of word x contributes weight $w(x, \phi)$. We extend the notation to represent the contribution of a Move operation on x as $w(x, x)$. Edit distance **ed1** can then be described as the following assignment of weights:

$$w(\phi, x) = 1$$

$$w(x, \phi) = 1$$

$$w(x, x) = 0$$

For convenience, we represent the situation where a word is in neither the source nor target revision as $w(\phi, \phi)$ and assign it a weight of zero.

We consider three revisions, just as we did in analyzing edit longevity in Fig-

ure 4.5: v_{k-1}, v_k, and v_j. To verify the triangle inequality,

$$d(k-1,k) + d(k,j) \geq d(k-1,j),$$

we enumerate the possible cases of word x existing in each revision:

1. word x exists in revision v_{k-1}, but not in v_k nor v_j. This translates into deletion operations for $d(k-1,k)$ and $d(k-1,j)$, so that we have

   $$w(x,\phi) + w(\phi,\phi) \geq w(x,\phi),$$

   which is true.

2. word x exists in revision v_k, but not in v_{k-1} nor v_j.

   $$w(\phi,x) + w(x,\phi) \geq w(\phi,\phi).$$

3. word x exists in revision v_j, but not in v_{k-1} nor v_k.

   $$w(\phi,\phi) + w(\phi,x) \geq w(\phi,x).$$

4. word x exists in revision v_{k-1} and v_k, but not in v_j.

   $$w(x,x) + w(x,\phi) \geq w(x,\phi).$$

5. word x exists in revision v_{k-1} and v_j, but not in v_k.

   $$w(x,\phi) + w(\phi,x) \geq w(x,x).$$

6. word x exists in revision v_k and v_j, but not in v_{k-1}.

$$w(\phi, x) + w(x, x) \geq w(\phi, x).$$

7. word x exists in all of revisions v_{k-1}, v_k, and v_j.

$$w(x, x) + w(x, x) \geq w(x, x).$$

All of these statements are true, so it would seem that edit distance **ed1** satisfies the triangle inequality. The complication arises in the requirement that there be a minimum number of words to be considered a Move operation. This restriction means that it is possible for a word to exist in both revisions, but be considered a Deletion and Insertion. For example, if word x exists in all three revisions, a different possible analysis is:

$$w(x, x) + w(x, x) \geq w(x, \phi) + w(\phi, x),$$

which does not hold true, and thus the triangle inequality sometimes will break down.

The proof that edit distance **ed2** always satisfies the triangle inequality follows a similar analysis, but using different weights:

$$w(\phi, \phi) = 0$$
$$w(\phi, x) = 1$$
$$w(x, \phi) = 1$$
$$w(x, x) = 1$$

These weights result in every statement holding true (even the alternative analysis when

a word match is lost because of the minimum words requirement) so that the full triangle inequality also holds true.

Examining the performance of **ed2** in Appendix F, we note that it was far from the best performing definition of edit distance in terms of predicting vandalism in the PAN-WVC-10 dataset. Although using an edit distance formulation that satisfies the triangle inequality is desirable, it is neither necessary nor sufficient to achieve good performance. The goal of an edit distance function in our context is to estimate the amount of *effort* that an author expends in creating an edit from one revision to another. We chose to follow a model which prefers the longest possible match between source and target revisions, but another viable route is to minimize the amount of text which is rearranged (i.e., minimize the number of transpositions) [109]. We leave as an open question how to best characterize the work that an author does.

4.7 Conclusions

We propose two measures of revision quality computed from Wikipedia's revision history. The measure *text longevity* is based on an intuitive model of computing the text added by authors at each revision and detecting how much of that text remains within the article in subsequent revisions; to account for the variation in the amount of preserved text over the subsequent revisions, we model the change as a geometrically decaying process and compute the decay rate as a single value to describe the variation. The measure *edit longevity* was developed to address the reality that authors also delete and rearrange text, and that these are valuable contributions to the Wikipedia. We use edit distance [58] to describe the amount of *effort* that an author puts into making a revision to an article; this is the basis for computing edit longevity, which estimates the amount of effort by an author that brings the article text closer to some future version

of the article.

We evaluate these two measures using the PAN-WVC-10 dataset, which is manually annotated to indicate which revisions are vandalism and which are well-intentioned edits, and treat each as a predictor of vandalism. We find that edit longevity performs much better than text longevity. Overall, these results are encouraging for using edit longevity and text longevity as signals for inferring the community feedback of an author's edit. Knowing the quality of edits, we can build an author reputation system upon these signals; we describe such a system in Chapter 6.

Chapter 5

Sizing Up Authors

5.1 Introduction

History is filled with examples of partnerships, people collaborating to create something larger than they could achieve on their own; but how do you value the contributions of the individuals? How do we say whether Apple benefited more from the technical designs of Steve Wozniak or the design intuition of Steve Jobs? The reality is that the very notion of what is a *contribution* depends on perspective and relative priorities of importance. When multiple authors work on a single book, how do you measure who did the work, and how should the revenue be split?

Today, online collaboration is growing by leaps and bounds, fostered under the name *Web 2.0*: "the activities of users generating content (in the form of ideas, text, videos, or pictures) could be *harnessed* to create value" [123]. In Silicon Valley, we hear examples every day of companies built atop the contributions of their users:

This chapter presents material previously pubished as [5].

Google, Facebook, Twitter, Flickr, to name a few famous ones. For the most part, these sites operate under a motto of "more is better," and only a few sites try to estimate a quality of contributions (e.g., Amazon and eBay). These sites are distinct from the Wikipedia, because although users generate content, and even collaborate in some sense, they don't actually work on the same content.

Within the Wikipedia community, there has been some discussion about measuring contributions [111, 102], in the context of whether some restrictions would improve the overall quality of the Wikipedia. Another motivation for understanding contributions by users is for attribution purposes: the Creative Commons license that the Wikipedia content is available under requires attribution of all authors, which is currently taken to include spammers and other vandals. When wikis [57] are used in a corporate setting, measuring contributions can also be a proxy for the productivity of workers.

In this chapter, we examine multiple ways that contributions can be defined within the Wikipedia. We explore the distribution of users under these different measures, and make some observations.

5.2 Related Work

From our point of view, measuring contributions to a collaborative work seems most like the software engineering practice of counting source lines of code to estimate programmer effort and productivity [91, 32]. There are several other productivity measures that have roots in the manufacturing process [28], such as measuring the number of defects or customer satisfaction [104].

The problem of measuring contributions to the Wikipedia seems to have first arisen in the context of trying to understand the process by which knowledge is accumulated and organized in such a large group collaboration (a discussion informed by such works

as [14, 9, 101, 84]). Wales conducted a survey in December 2004 which finds that half the edits within the Wikipedia are made by only 2.5% of logged in users [111]. Swartz challenged back that *edit counts* capture only one part of the story; counting the size of edits presents a different picture, and this has policy implications for the Wikipedia in how it decides to encourage more contributions [102].

In trying to ascribe a source to the many contributions that make up the Wikipedia, Anthony et al. measure the *survivability* of an edit by looking at the percentage of characters retained in later edits [7]. The authors use the entire content of the version being evaluated (since the author could have made edits to any part of the content), which distinguishes their measure from those we developed in Chapter 4 that are designed to track only the changes done specifically by the author being evaluated. Given that caveat, they find that "Good Samaritans" (one-time anonymous users) have the highest quality contributions overall.

Kittur et al. take up the question of whether *elite* or *common* users contribute more content, analyzing both the number of edits made by authors and the total size of the edit differences [54]. Their data suggests that both measures point to the same conclusion: that elites dominated content-generation in the early history of the Wikipedia, but the workload had shifted to the common users by mid-2006. The opposite conclusion is reached by Ortega et al., who revisit the question of how contributions (as measured by edit counts) are distributed over the Wikipedia user base and use Gini coefficients to quantify the concentration of core contributors [75].

Many other works exist that are, in the abstract, considering author contributions. In practice, we find that most works use edit count as their contribution measure [125, 12, 100, 74, 98, 76]. We suspect that this is due to the relative simplicity of counting edits versus computing text differences.

Our own work differs from previous research in two important ways. First, we propose that there are multiple measures that can represent the "size" of a contribution. Second, and more importantly, we observe that not all contributions of the same size have equivalent "quality." As an example, consider the addition of the sentence "UC Santa Cruz rocks your socks" to an article. This contribution of six words is considered equivalent to other legitimate contributions, but actually is a *negative contribution* in that it creates more work for some other author (who must delete it). We explore variations of both size and quality.

5.3 Primitives

In order to construct functions that compute the contributions of authors, we need some primitives to work with. Our view is that any sizing up of the work or productivity of Wikipedia authors must take into account the fact that not all contributions are positive; approximately 7% of edits are vandalism [79, 78]. We propose that besides measuring the magnitude of work done by an author, to gauge the productivity of authors towards the goal of producing a useful reference requires tempering that magnitude with a quality measure. One way to achieve this is to factor the two together as "*quantity* · *quality*," with the following desirable property for quality: it should be signed, where positive values are towards a common goal and negative values are away from that goal.

We define various measures of author contribution, taking into account the amount of text added or edits performed by the author and the quality of those changes. We would like to measure contributions both in absolute terms, as the amount of text that was added by an author or the amount of edits made by an author, and in relative terms, where we take into account the quality of the edits. The contributions of all authors is cumulative over the entire revision history of the Wikipedia; for our experiments, we

picked revisions of all articles previous to October 1, 2006.

For every article, $a \in \mathbb{A}$, in the Wikipedia, we consider each version $v_i \in \mathbb{V}[a]$ to be edited by the author $A_i = \mathrm{RevAuthor}(v_i)$. Each of the subsequent authors $A_{i+1}, A_{i+2}, \ldots$ can either retain, or remove, the changes performed by A_i in bringing version v_{i-1} to v_i. These authors who edit article a after v_i are implicitly providing feedback on the content of v_i, and hence act as *judges* of the contribution made by author A_i. We therefore define $J_a^n(i)$ to return the set of up to n **next** revisions after v_i, such that the author of each $v \in J_a^n(i)$ is restricted to $\mathrm{RevAuthor}(v) \neq \mathrm{RevAuthor}(v_i)$. It should be noted that other formulations of $J_a^n(i)$ are possible, such as the one proposed in [17] where only high-reputation authors are selected as judges.

Recalling that

$$\mathbb{V}[a] = [v_1, v_2, \ldots, v_n],$$

we define $\mathrm{RevPos}(v_i) = i$; note that a is implicit in v_i, since a revision is part of the history of some specific article. Crucially, to define contribution formulas for each user, we also define the map

$$\mathbb{E} : \mathbb{U} \times \mathbb{A} \to 2^{\mathbb{V}},$$

which, given a user $u \in \mathbb{U}$ and an article $a \in \mathbb{A}$, returns the set of revisions that were created by user u for article a.

Quantity Measures.

We would like to measure the *size* of an author's contribution when they create version v_i of some article $a \in \mathbb{A}$. The most obvious quantity that can be measured is counting how many words the author added in version v_i, which can be computed from the text difference in going from v_{i-1} to v_i. If we let $I(v_{i-1}, v_i)$ represent the number of words

that were inserted in going to version v_i, our first quantity measure to measure the size of a *text contribution* is:

$$txt(v_i) = TSurv_a(i,i) = I(v_{i-1}, v_i),$$

which is a specific case of the more general definition in Equation 4.1.

As we have previously noted, counting only the words added in an edit ignores the fact that some users of Wikipedia do maintenance work in the form of rearranging text or removing vandalism. To capture this extra behavior, we would like to also measure what words were deleted in the edit, $v_{i-1} \rightsquigarrow v_i$, as well as how many words were rearranged. This is known in the literature as the *edit distance* between versions v_{i-1} and v_i. There are several ways to compute edit distance [58, 105], usually based on insertions and deletions of characters.[1] Our formulation is instead based on words as the fundamental unit, to more closely approximate how people perceive edits. We define the edit distance in terms of the following quantities:

- $I(v_{i-1}, v_i)$ is the number of words that are inserted,
- $D(v_{i-1}, v_i)$ is the number of words that are deleted, and
- $M(v_{i-1}, v_i)$ is the number of words that are moved, times the fraction of the document that they move across,

which can be computed according to the text differencing algorithm described in Chapter 3. The size of the *edit contribution* for v_i is then given by:

$$d(v_i) = d_a(i-1, i) = \max(I, D) - \frac{1}{2}\min(I, D) + M,$$

[1]The careful reader will observe that the definition of $txt(v_i)$ is actually an edit distance as well, albeit one which ignores deletions and text rearrangement.

which is a specialization of the more general edit distance in Equation 4.3.

Quality Measures.

In addition to the quantity measures defined above, we need multiple quality measures to choose from. We start with the two quality measures derived in Chapter 4:

- $q^{10}_{tdecay}(i)$ is the value that best describes the text survival of text inserted in version v_i over the next ten revision as an exponential decay, defined in Section 4.3. This value ranges from 0 for completely removed text, to 1 for text which is completely preserved.

- $q^{10}_{elong}(i)$ is the average edit longevity of the contribution of version v_i as judged by up to ten judges in the immediate future. This value ranges from -1 for completely reverted edits, to $+1$ for completely preserved edits. See Section 4.4 for the development of this measure.

A third way to measure the quality of a text contribution is to simply sum the fraction of text that remains over the succeeding ten revisions:

$$q^{10}_{tsurv}(i) = \frac{1}{TSurv(i,i)} \cdot \left(\sum_{v \in J^{10}(v_i)} TSurv(i, \mathrm{RevPos}(v)) \right)$$

This value generally ranges from 0 for text which is immediately removed, to $+10$ for text which completely survives all ten revisions.

5.4 Contribution Measures

We now present several potential contribution measures in the form of "*quality · quantity*," using the building blocks described in the last section.

Number of Edits

The simplest quantitative measure of contribution for authors is to compute the number of revisions they authored. In previous works, this is referred to as the *number of edits* made by an author [111, 125, 54, 98], or simply the *edit count*. We define this precisely for some user $u \in \mathbb{U}$ as:

$$\mathsf{NumEdits}(u) = \sum_{a \in \mathbb{A}} \sum_{v \in \mathbb{E}(u,a)} 1 \cdot 1.$$

Text Only

Another very natural measure of author contribution is to count up how many words were added by each author, during the course of all their revisions. Since there is no quality measure involved, we refer to this measure as TextOnly, and define it for each $u \in \mathbb{U}$ as:

$$\mathsf{TextOnly}(u) = \sum_{a \in \mathbb{A}} \sum_{v \in \mathbb{E}(u,a)} 1 \cdot \mathit{txt}(v).$$

We refer to this measure as the *absolute* text contribution measure.

Edit Only

Correcting grammar, polishing the structure of article, and reverting vandalism are all chores [12] which must be done to keep the Wikipedia presentable. We note that measuring the size of the change in each revision is able to reward both authors who write new text, as well as authors who polish existing text. To achieve thisTo achieve this, weee measure the edit distance between the version v_i being evaluated and version v_{i-1} that immediately preceeded it as the size of the change. The EditOnly measure is thus

defined for all $u \in \mathbb{U}$ as:

$$\mathsf{EditOnly}(u) = \sum_{a \in \mathbb{A}} \sum_{v \in \mathbb{E}(u,a)} 1 \cdot d(v).$$

We refer to this measure as the *absolute* edit contribution measure.

Text Longevity

The next level of sophistication is to incorporate non-constant quality measures into the calculation of contribution. We desire the text longevity of a revision to be the amount of original text that was added by the author $\mathrm{RevAuthor}(v_i)$ for a revision v_i, discounted by the text quality measure $q^{10}_{tdecay}(v_i)$, which describes how the text decays over the next several revisions.

$$\mathsf{TextLongevity}(u) = \sum_{a \in \mathbb{A}} \sum_{v \in \mathbb{E}(u,a)} q^{10}_{tdecay}(v) \cdot txt(v)$$

Edit Longevity

Similar to the text longevity measure, we define the edit longevity of a revision v_i as the edit contribution, discounted by the average edit quality measure $q^{10}_{elong}(v_i)$. As with all the measures, we accumulate contributions based on edit longevity over all revisions edited by each user $u \in \mathbb{U}$:

$$\mathsf{EditLongevity}(u) = \sum_{a \in \mathbb{A}} \sum_{v \in \mathbb{E}(u,a)} q^{10}_{elong}(v) \cdot d(v)$$

Ten Revisions

A simpler method for measuring how useful newly inserted text is, is to simply add up how many words survive over the next ten revisions. Large contributions are thus richly rewarded, if they survive; smaller contributions have a slightly better chance of surviving for the entire ten revisions, thus encouraging change — but not too much change.

We consider the ten revisions that follow any revision v_i of an article, and accumulate the amount of text contribution that was made in v_i that remained in each of those ten subsequent revisions of the article. We call this measure TenRevisions and define it for each $u \in \mathbb{U}$ as follows:

$$\mathsf{TenRevisions}(u) = \sum_{a \in \mathbb{A}} \sum_{v \in \mathbb{E}(u,a)} q^{10}_{tsurv}(v) \cdot txt(v).$$

Text Longevity with Penalty

A last variation that we propose is to combine text longevity with edit longevity in such a way that authors of new content are rewarded, but vandals are actively punished for both inserting and deleting text. Text longevity, as we have defined it, already does not reward vandals — vandals either insert no text, or the text they insert is immediately removed; both cases result in a text longevity of zero for the revision. Vandals are still able to accumulate positive contributions from other revisions, however, while disrupting other authors with their vandalism. To counteract this, we can punish vandals when their work is reverted by referring to when the edit longevity quality is negative.

This leads to the following definition of our punishing measure for every $u \in \mathbb{U}$:

$$\mathsf{TextLongevityWithPenalty}(u) =$$
$$\mathsf{TextLongevity}(u) + \sum_{a \in \mathbb{A}} \sum_{v \in \mathbb{E}(u,a)} \min(0, q^{10}_{elong}(v)) \cdot d(v).$$

5.5 Implementation

As part of our research into author reputation and text trust [2, 1], we have created a modular tool for processing XML dumps from the Wikipedia. It analyzes all the revisions of a page, filtering down the revisions to remove consecutive edits by the same author, and computing differences between revisions to track the author of each word and measure how the author might have rearranged the page. These results can be passed to any of several modules to do additional processing; we use the tool to reduce the enormous collection of data down to a much smaller *statistics file*. We process the statistics file with a second tool, which we instrumented to calculate the various contribution measures we have defined.

Our analysis is based on main namespace (`NS_MAIN`) article revisions from the Wikipedia dump of February 6, 2007, which we process to create a reduced statistics file. The statistics file contains information about every version, including the amount of text added, the edit distance from the previous version, and information about how the edit persists for ten revisions into the future. To ensure that each version we considered had revisions after it, we consider only versions before October 1, 2006. After further processing on the file, we used R [83], an open source statistics package, to analyze the resulting data.

Bots. During the course of our analysis, we found that some authors were extraor-

dinary outliers for multiple measures. Some investigation into the most extreme cases revealed that bots were making automated edits to the Wikipedia, and that a few bots dwarfed manual labor in the edit based measures EditLongevity and EditOnly. We also found that there are bots that improve content, and bots that vandalize it. We chose to identify bots as those with a username which ends in the string "bot;" While this does not include every bot (especially the ones that vandalize), it is a useful first approximation. We found 614 bots in total as of October 1, 2006.

Vandals. There is a similar problem in trying to define vandals, since such authors don't register themselves as such. For our purposes, we decided to define a vandal as someone who, on average, makes an edit which is completely reverted. Precisely, we define a vandal who meets one of two criteria: $q^{10}_{tdecay} < 0.05$, or $q^{10}_{elong} < -0.9$. We justify this choice in the next section.

5.6 Analysis

We begin our analysis with some information about the data we are analyzing. Our reduced statistics file includes over 25 million revision records. Figures 5.1 and 5.2 were created by drawing a random sample of 5 million records, due to memory limitations of the software package.

In Figure 5.1, we show the frequency distribution of the two quality measures q^{10}_{tdecay} and q^{10}_{elong} over the revisions we sampled. We see both measures are heavily biased towards $+1$, indicating that most revisions to the Wikipedia are generally considered useful by succeeding authors. This confirms the intuition that more "good people" than "bad people" must contribute, otherwise the Wikipedia would have a difficult time maintaining the community which continues to extend the online encyclopedia in a useful way.

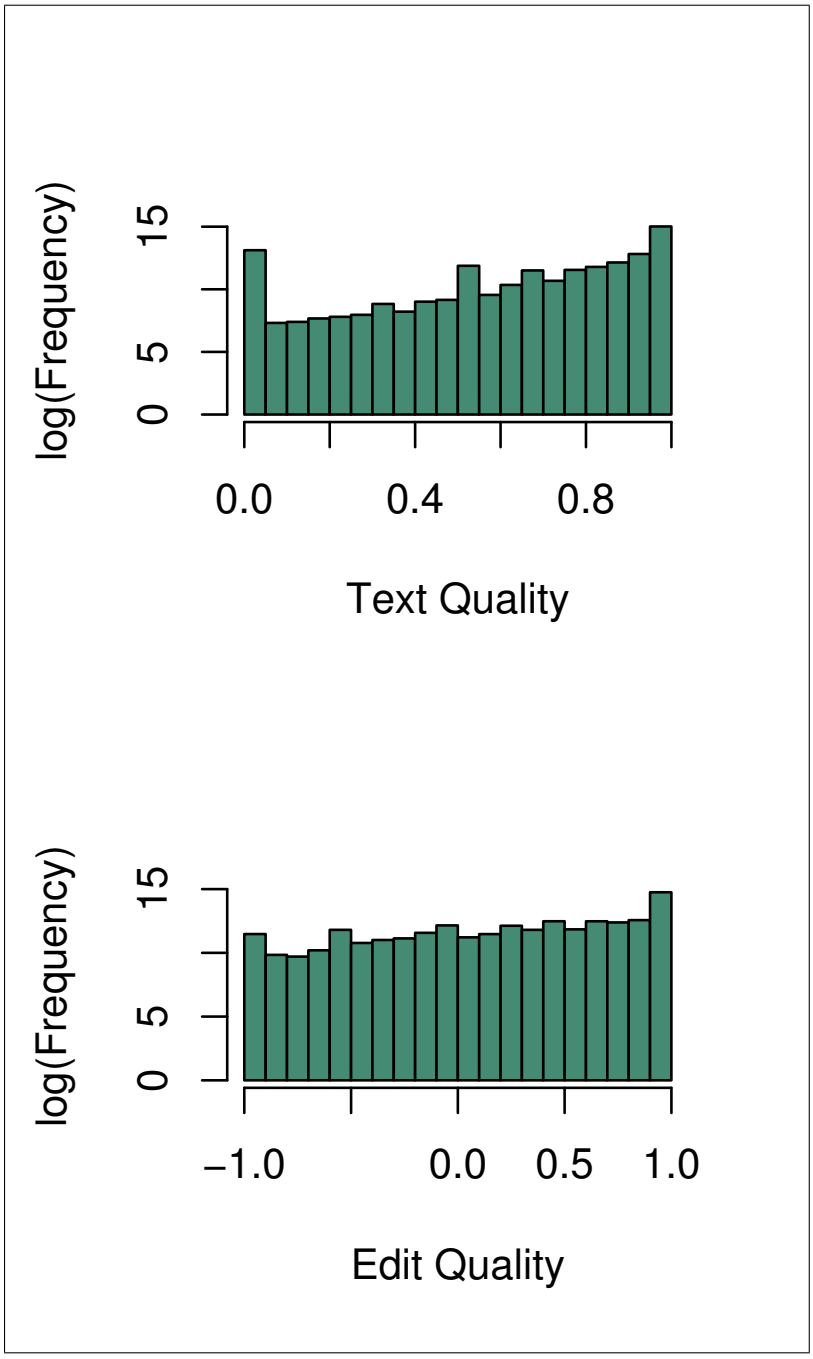


Figure 5.1: This graph shows the text quality q^{10}_{tdecay} and edit quality measure q^{10}_{elong} for 5 million randomly selected records of each type.

Delving directly into the data for text quality, we observe that 10% of the revisions made had $q^{10}_{tdecay} \leq 0.05$ while 66.67% of the revisions had $q^{10}_{tdecay} > 0.95$. When $q^{10}_{tdecay} = 0$, the text is immediately deleted in the next revision, so we can infer that these revisions are the work of vandals. When we look at the size of contributions made, we noticed that 6% of the amount of new text added had $q^{10}_{tdecay} = 0$, whereas 76.21% of the new text added had $q^{10}_{tdecay} > 0.95$. From this we conclude that authors mostly add good new text.

The data is less stark for edit quality. When we looked at revisions, we saw that 1.9% of the revisions had $q^{10}_{elong} \leq -0.9$, whereas 51.12% had $q^{10}_{elong} > 0.9$. In fact, 84.71% of revisions had positive edit quality. When taking the size of each edit (the edit contribution) into account, we noticed that 7.5% of the edit contributions had $q^{10}_{elong} \leq -0.9$, whereas 61.39% had $q^{10}_{elong} > 0.9$. Moreover, 1.6% of the edit contributions were immediately reverted. From these statistics, we conclude that authors mostly do good edits, but that contributions are massaged a bit by later editors.

Figure 5.2 shows the absolute text and edit contributions, $txt(v_i)$ and $d(v_i)$, for the sets of sampled revisions. It is important to note that these two graphs are using the logarithm of the size of contribution, along the x-axis; edit sizes can fall below $+1$, due to the way we compute edit distance. For moved words, they are included as a fraction of how much of the document they move across; if words are replaced with an equivalent number of words (as can happen with synonyms replaced for clarity), the net contribution to edit distance is zero. Thus, the frequency count for edit sizes between 0 ad 1 suggests that a good fraction of revisions involve rearranging of text. Beyond that, we can conclude that contributions, as measured by text added or by edit distance, are predominantly under 100 words.

In Figure 5.3 we show the average edit quality and average text quality for all non-

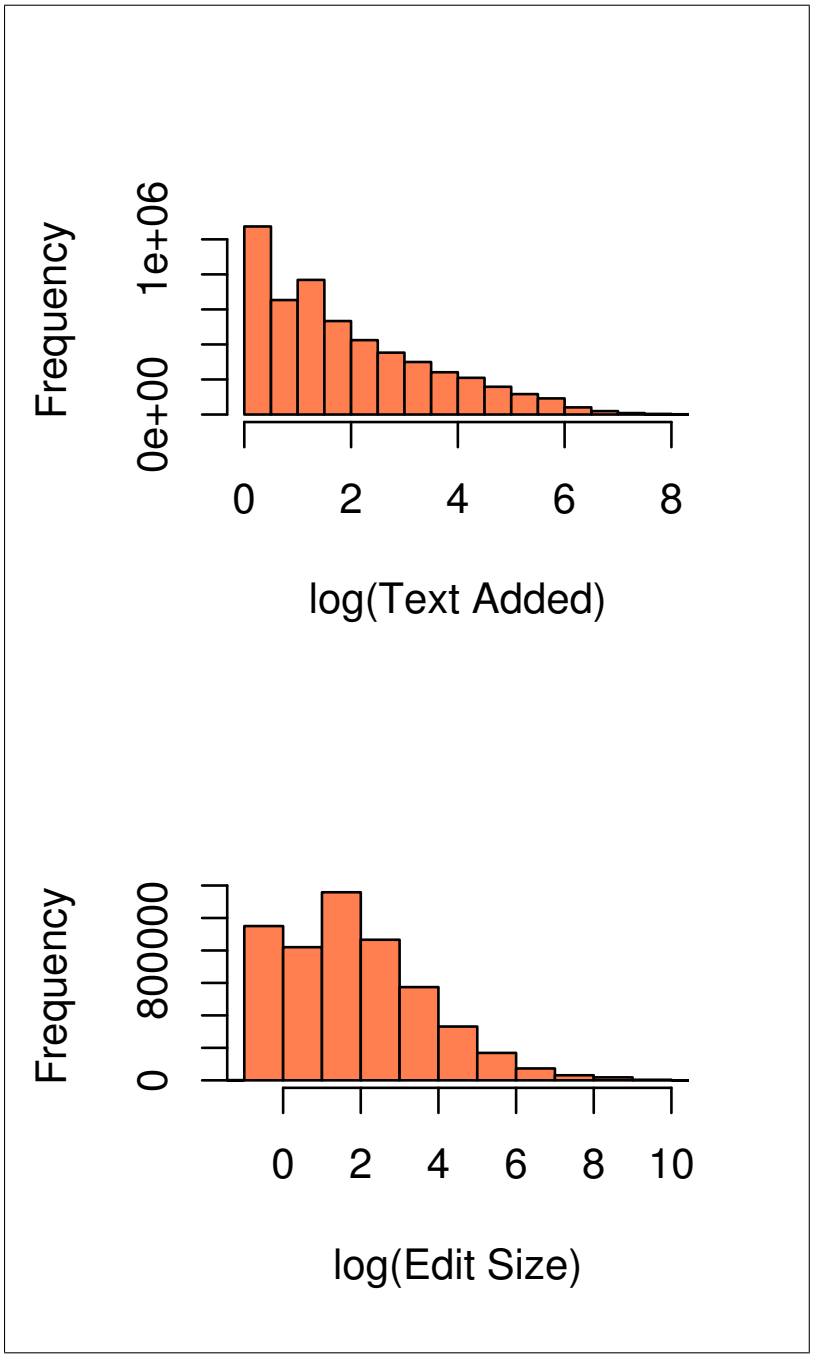


Figure 5.2: This graph shows the absolute text and edit contributions on a log scale, for 5 million randomly selected records of each type.

anonymous authors. In order to compute this, we took all revisions created by each author and took an average of the text and edit qualities of those revisions. We notice that 15.9% of authors had $q^{10}_{tdecay} \leq 0.05$ and 6.3% of authors had $q^{10}_{elong} \leq -0.9$. These are shown by the bars on the left extreme of the histograms in Figure 5.3. This sharp increase in the number of authors at the lowest end of our quality measures, combined with our previous analysis of revisions and contributions with respect to quality, gives us some justification to define vandals as those authors who have either $q^{10}_{tdecay} \leq 0.05$ or $q^{10}_{elong} \leq -0.9$ on average. The identification of vandals can be made more precise using more sophisticated analysis of our data, as is done in Chapter 7.

During our investigations comparing the proposed measures, we found an unusually large fraction of non-anonymous authors having scores relatively close to zero. This suggested that many users had made a relatively small number of revisions, and that the absolute text and edit contributions of the revisions tended to be small, or that the quality tended towards zero. This is consistent with the power law distribution for edits per author (Lotka's law) detected by [108]; we confirmed the distribution for our data (shown in Figure 5.4) and observed that 362,461 authors made only one edit: over 46% of the total 777,223 authors we tracked. In Figure 5.5 we show the edit quality measure for these authors. In contrast to the edit quality distribution over all authors from Figure 5.1, we notice that the edit quality for these authors are almost evenly distributed across the entire quality range (except for the two extreme values).

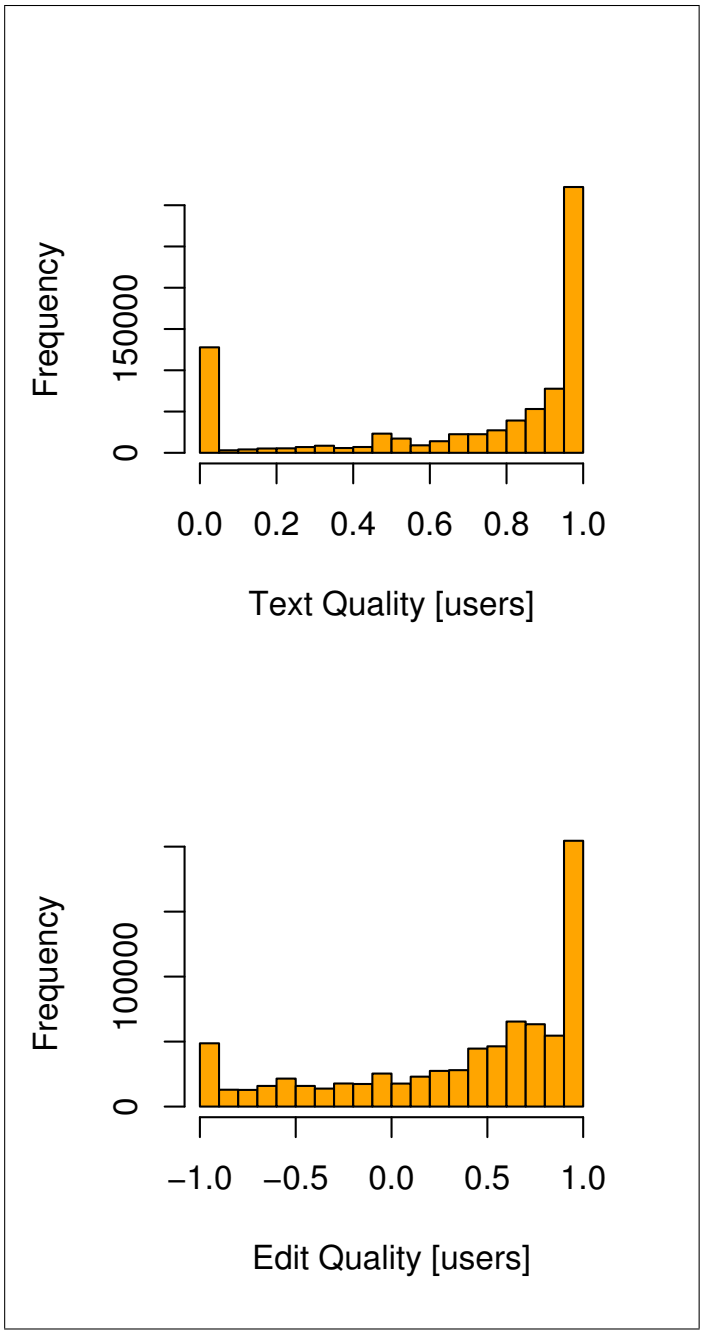


Figure 5.3: This graph shows the average text quality q^{10}_{tdecay} and the average edit quality measure q^{10}_{elong} over all non-anonymous authors.

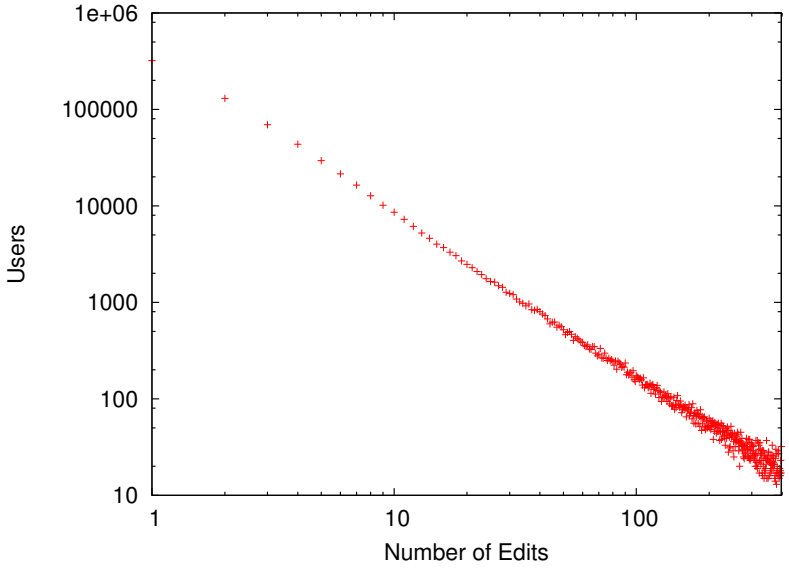


Figure 5.4: The distribution of the number of edits that each author made. Over 46% of the non-anonymous authors make a single edit in the main English Wikipedia.

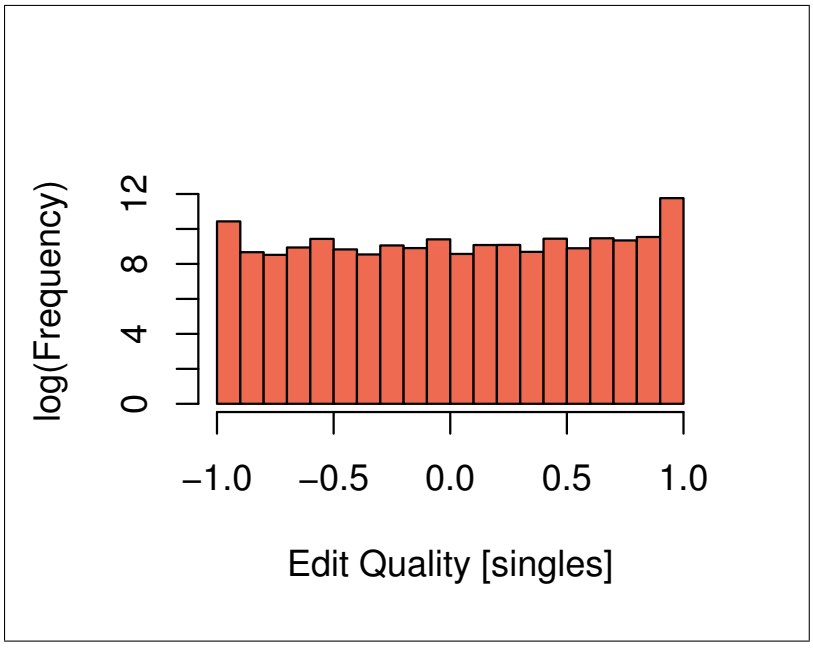


Figure 5.5: This plot shows the q^{10}_{elong} of the non-anonymous authors who made only a single edit.

Comparing Measures

We next present the correlations between the various measures in Table 5.1. These are correlations with respect to the amount of contributions made by all non-anonymous authors, excluding those we've classified as vandals. From the correlation table, we notice that text based measures are better positively correlated with each other. Similarly, the edit based measures are better positively correlated with each other as we expected. The measures EditLongevity and EditOnly are highly correlated as borne out by the fact that a large percentage of the edits are of good quality. We notice that the same is true for TextLongevity and TextOnly. The correlation between TextLongevityWithPenalty and the absolute measure EditOnly is low, demonstrating that TextLongevityWithPenalty penalizes authors for bad edits, gives no credit to good edits, and accumulates the quality discounted text contribution measure TextLongevity. Therefore, authors need to contribute high quality text, while ensuring that they have no bad edits to get a high score on TextLongevityWithPenalty. TenRevisions being a text contribution measure, is highly correlated with the other text contribution measures TextOnly and TextLongevity. NumEdits is positively correlated with all measures as we would expect, since the majority of contributions are deemed good by each of the quality measures.

While TextOnly and EditOnly appear to be reasonable measures of author contribution, we have found evidence that vandals accrue large contributions against these measures. For instance, we found that author 1065172 is at the 99^{th} percentile when measured using TextOnly, but is nearly at the bottom of the ranks, at 0.000001 percentile when we look at his TextLongevityWithPenalty measure. We found five revisions in which this author added new text, but four of those were immediately reverted. The only revision that was kept around was a one word addition to a page! From the ed-

$Measures$	$EditLong$	$EditOnly$	$NumEdits$	$TenRevs$	$TextLong$	$TextOnly$	$TextWPen$
$EditLong$	1.000	0.999	0.28	0.070	0.075	0.16	-0.32
$EditOnly$	0.999	1.000	0.29	0.071	0.077	0.16	-0.33
$NumEdits$	0.283	0.286	1.00	0.361	0.417	0.45	0.27
$TenRevs$	0.070	0.071	0.36	1.000	0.983	0.96	0.89
$TextLong$	0.075	0.077	0.42	0.983	1.000	0.98	0.90
$TextOnly$	0.158	0.164	0.45	0.963	0.983	1.00	0.82
$TextWPen$	-0.320	-0.326	0.27	0.886	0.897	0.82	1.00

Table 5.1: This table gives the pairwise correlations of the different measures we have defined in this chapter, examining only non-anonymous users that are not classified as vandals.

its made by this author, we saw that he is a spammer. On the other hand, the author was below the 25$^{\text{th}}$ percentile when measured by TextLongevity. Using the EditLongevity measure, this author was below the 0.001 percentile; among the lowest in rank. Therefore, we argue that the measures that discount TextOnly and EditOnly by a text or edit quality measure are more indicative of the "useful" work added to the Wikipedia. We argue that NumEdits is not as good a measure, since vandals and bots can easily make large numbers of bad edits.

We present two figures, Figure 5.6 and Figure 5.7, which have been restricted to a region containing the bulk of the data points. In Figure 5.6, we see a vee shape, which separates the authors into two groups: those that have positive edit quality and those that have negative edit quality, as measured by q^{10}_{elong}. The worse the quality of edits made by authors the less they accumulate of the EditLongevity measure, whereas the EditOnly measure, being oblivious to edit quality, attributes the same contribution to an author whose contributions persists as it does to an author whose contributions do not. On the negative side of EditLongevity, there are points that represent vandals, who edit large sections of existing pages, which are then immediately reverted. Clearly, EditOnly ranks some of these authors very highly, whereas EditLongevity is able to distinguish them and rank them very low.

In Figure 5.7, we see a similar vee shape; in this case, TextLongevity cannot go below zero as the text quality measure is always non-negative, so vandals, by our definition, receive no contribution. As before, the measure that incorporates quality can distinguish vandals from non-vandals and attribute a contribution measure to authors that is proportional to the merit of their contribution.

Of the various measures we introduced, TextLongevityWithPenalty is perhaps the one with the least tolerance, since by this measure, the only way an author can ac-

cumulate contribution is by adding new text that persists and by making edits that are judged to be of good quality. Further, this measure does not reward authors for good edits, but penalizes them for bad edits. In Figure 5.8, we plot TextOnly against TextLongevityWithPenalty. We see the vee shape, with vandals falling on a noticeable line in the fourth quadrant, that has no TextOnly contribution. Since almost all new text added by vandals is immediate reverted, and their edits always have low quality, we notice that they get low negative TextLongevityWithPenalty contributions. In fact, we noticed that the bottom ten authors by rank when measured according to TextLongevityWithPenalty were all vandals with the exception of *AntiVandalBot*. We explain this in the subsection on bots.

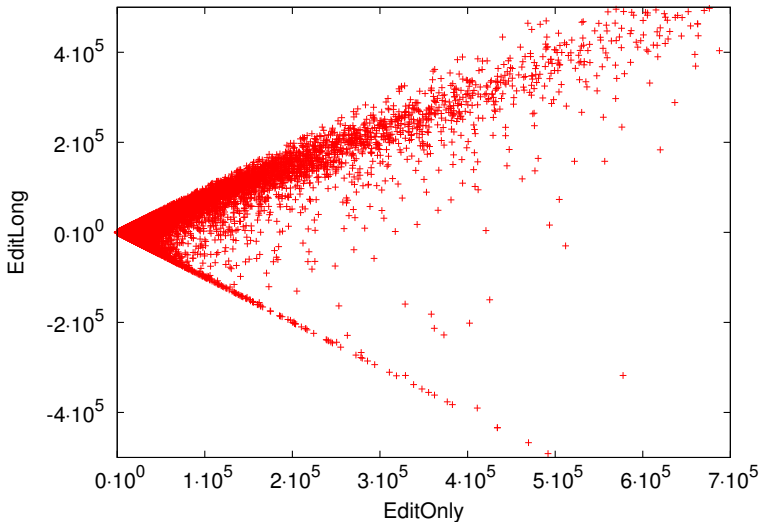


Figure 5.6: Comparing the absolute edit contribution, EditOnly, of a user with the edit longevity, EditLongevity. Notice that authors who are "all bad" are easily identifiable – and sometimes quite prolific.

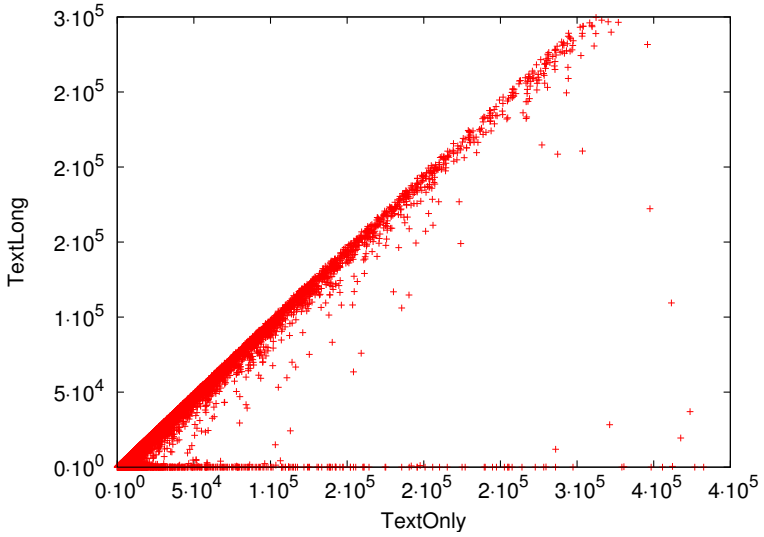


Figure 5.7: Comparing the absolute text contribution, TextOnly, with the contribution as measured by text longevity, TextLongevity. We see that large contributors are either "all bad" or nearly "all good."

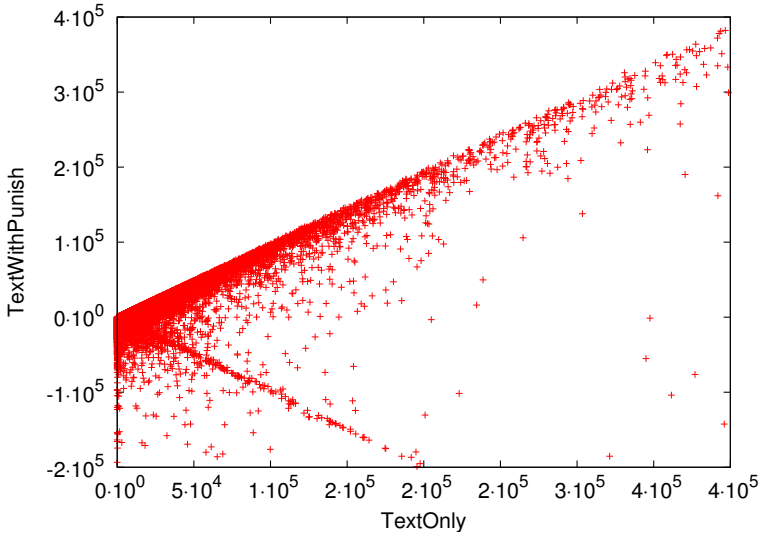


Figure 5.8: Comparing the absolute text contribution of an author, TextOnly, with their contribution as measured by TextLongevityWithPenalty.

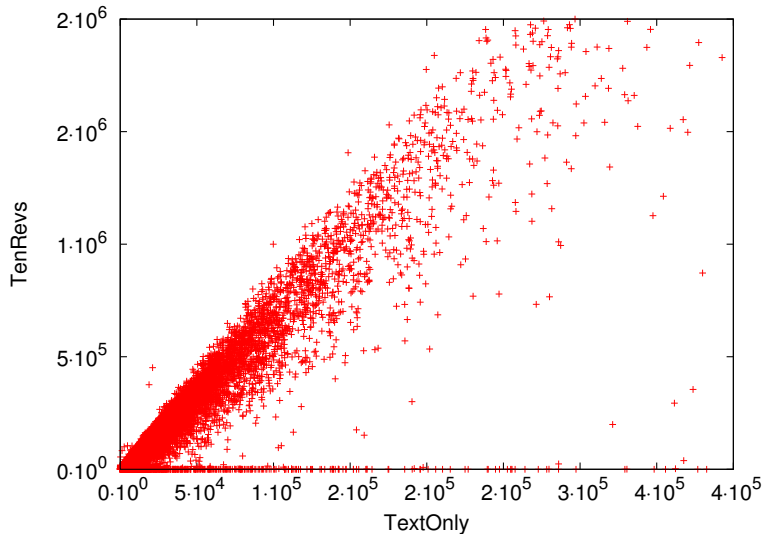


Figure 5.9: This graph compares how much text is initially added by a user (along the x-axis), with how much of the text survives over the next ten filtered revisions (along the y-axis). The higher up the y-axis a point is, the more text that survived all ten revisions. Most authors add under 100,000 words, and about half of what they add survives.

Ranking Authors

A different direction we explored was how these different measures end up ranking different authors. Since the contribution measures varied over such a wide range of values, with most people within a smaller region around zero, we hoped that ranking the authors would give us better insight into how the measures differed.

To this end, we computed the percentile rank (rounded up to the next even value for clarity in the image) of all non-anonymous authors, including those that we had classified as vandals, and then plotted them in 3-dimensional histograms; see Figures 5.10 and 5.11. An important point to remember about Figures 5.10 and 5.11 is that the low-lying regions of the graph are rarely zero — there are roughly between one and ten authors at each intersection, but this is so small compared to the areas that correlate

that we cannot see it on the graph. Both figures show a high degree of correlation that wasn't evident from the correlation scores in Table 5.1. Figure 5.10 shows that TextLongevity and EditLongevity generally agree in the ranking of users, except for the lowest scorers of TextLongevity. The lowest scorers of TextLongevity all receive a score of zero, but the "fence" seen in the figure is an indication of the fact that there are an enormous number of users which TextLongevity ranks equivalently but EditLongevity is able to further distinguish between. By contract, Figure 5.11 shows that TextLongevityWithPenalty roughly agrees with EditLongevity for all users except for a thin branch that score zero under TextLongevityWithPenalty but get a positive score under EditLongevity. This thin branch represents the group of users which do not add text, but instead only rearrange it or delete vandalism.

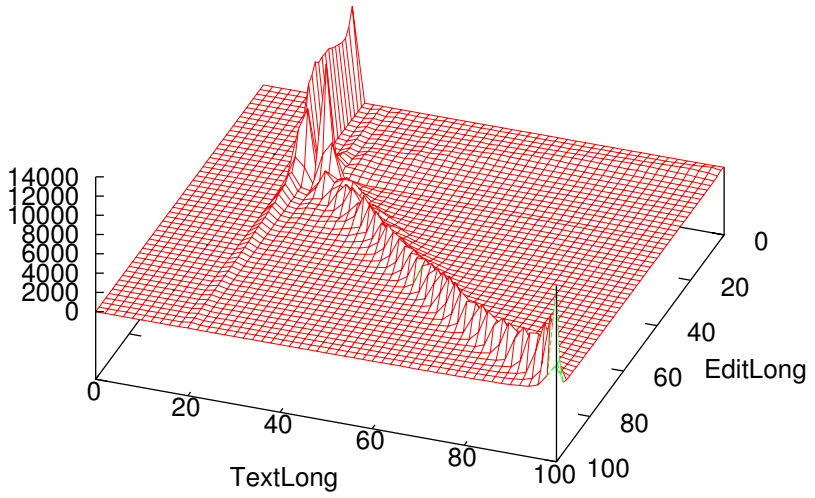


Figure 5.10: EditLongevity vs TextLongevity

We also include a 3-dimensional histogram comparing the percentile rankings as

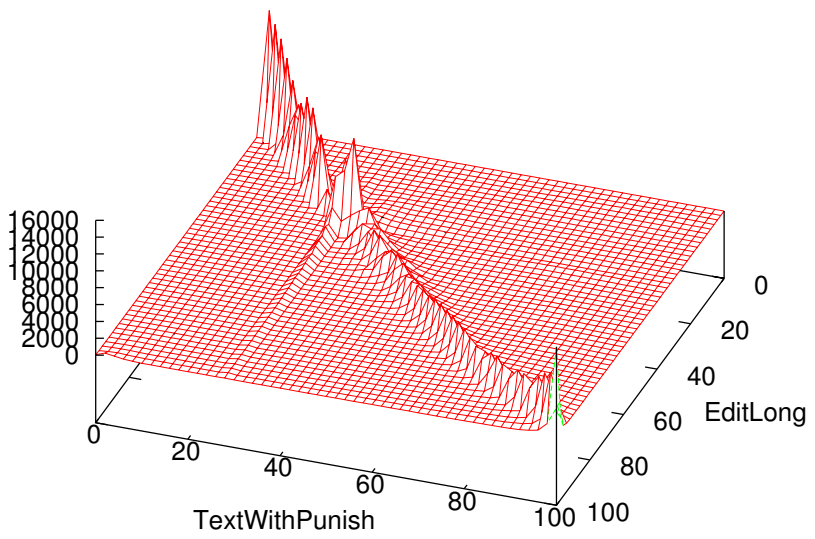


Figure 5.11: EditLongevity vs TextLongevityWithPenalty

determined by EditLongevity and NumEdits, in Figure 5.12. The "rows of fences" we see in Figure 5.12 are due to the large number of authors who make only a handful of edits; the NumEdits measure neither distinguishes them from each other, nor is it capable of distinguishing good contributions from bad contributions. This last point is important, that even users in the lowest percentile of EditLongevity can be rated very highly by NumEdits— demonstrating that it is much easier to game the NumEdits measure to achieve a high rank, while doing bad work.

Bot Behavior

There are several bots operating on the contents of the Wikipedia. Many bots are sanctioned by the community, and do useful chores such as automatically removing text which is likely to be vandalism, correcting spelling, and adding geographical

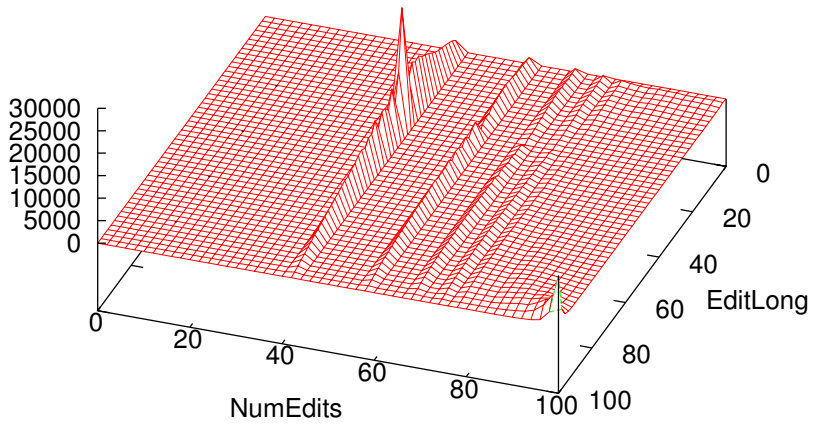


Figure 5.12: EditLongevity vs NumEdits

data. There are also bots which are created to vandalize pages, and sometimes well-intentioned bots run amock and accidentally vandalize pages as well. During the course of comparing the various contribution measures with each other, we found several bots (both good and bad) which were obvious outliers in the data. To analyze bots as a group, we selected all users which included the "Bot" moniker in their username; this self-identification does include some malicious bots, but obviously favors selection of good bots.

The edit and text quality measures for all bots are similar to that of all authors shown in Figure 5.1. We noticed that bots create a large number of revisions with high quality. We found that 69.56% of the revisions made by bots have a text quality measure of $q^{10}_{tdecay} > 0.95$. The percentage of revisions made by bots with $q^{10}_{tdecay} \leq 0.05$ was 9.2%. We found that 66.92% of the new text added by bots were with $q^{10}_{tdecay} > 0.95$ and

14.14% of the new text added by bots were with $q^{10}_{tdecay} = 0$, which means they were immediately reverted. Similarly, on the edit contributions of bots we found that 54.42% of the revisions with edits made by bots were of high edit quality, with $q^{10}_{elong} > 0.9$. The number of revisions having $q^{10}_{elong} < -0.9$ being negligible, only 1% by our analysis. When we counted all edit revisions that had a negative edit quality we saw that 12.73% of the revisions were judged to be of poor quality with $q^{10}_{elong} < 0$. We found that 93.3% of the edit contributions made by bots had positive edit quality and the remaining 6.4% had negative edit quality. As we would expect from a selection of bots which is biased towards good bots, 65.20% of the edit contributions made by bots had $q^{10}_{elong} > 0.9$, indicating that the work is generally regarded as high quality. The contributions with $q^{10}_{elong} < -0.9$ are 1.8%. This indicates that a large part of the text additions made by bots and a large part of the edit contributions made by bots survive indefinitely.

Furthermore, our analysis indicates that bots make large amounts of edit contributions compared to text contributions; the ratio of the size of edits EditOnly to the size of new text TextOnly for all bots is 11.61. Since the penalizing measure TextLongevityWithPenalty does not credit authors for good edits but reduces their TextLongevity contributions, by the amount of their bad edits as measured by EditLongevity, we notice that edits judged as being of poor quality overwhelm the smaller text contributions of bots in general, and *AntiVandalBot* in particular, resulting in a small overall contribution. We also note here that *SmackBot* did much better on this measure. *SmackBot* contributes more text than *AntiVandalBot*. Most of its edits are of smaller size than *AntiVandalBot*. Since they have similar quality measures, *AntiVandalBot* ends up with a lower score on TextLongevityWithPenalty when compared to *SmackBot*.

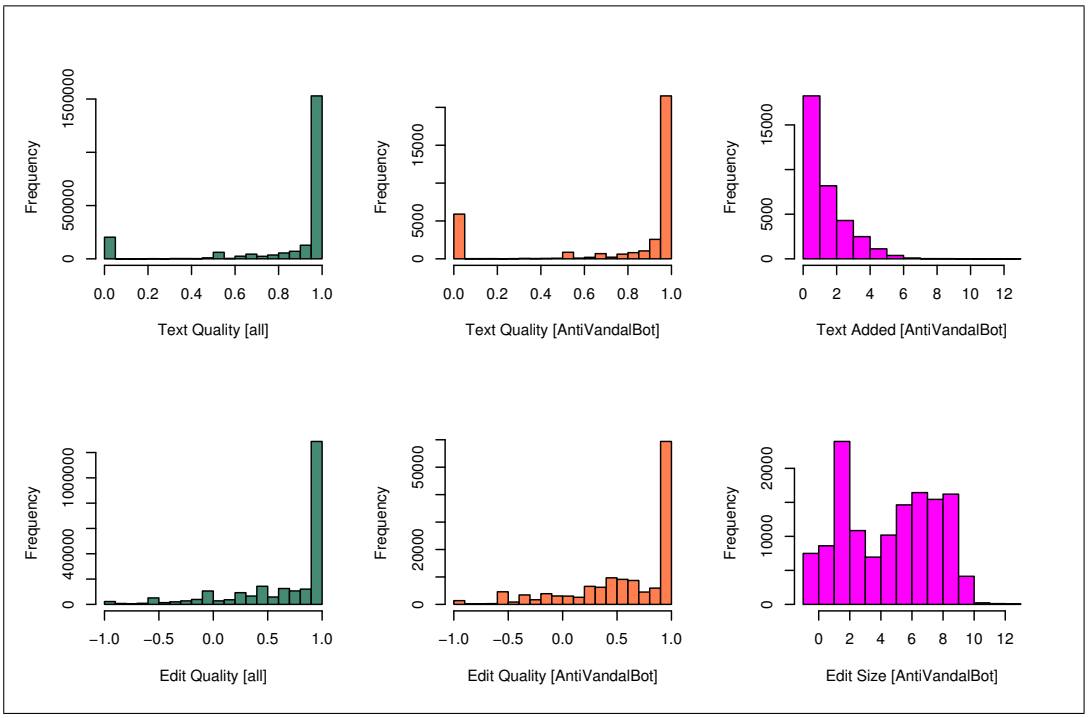


Figure 5.13: This graph shows the edit quality measure q^{10}_{elong} and text quality q^{10}_{tdecay} for all bots. A quality measure of 1 indicates that all the changes were preserved. A quality measure of 0 for text quality (and -1 for edit quality) indicates that all the changes were reverted in the very next revision. The histograms on the left are for all bots. The ones on the right are the quality measures and the absolute amounts of text and edit contributions for *AntiVandalBot*.

Sources of Error

Since we use filtered revisions, namely we collapse all consecutive revisions by the same author, and since we treat all anonymous authors identically, consecutive edits made by anonymous authors cannot be distinguished. We therefore discard all anonymous authors from our analysis: in any case, we are not measuring their contributions, as they cannot be individually attributed. We have noticed that there are anonymous authors who do good work on the Wikipedia, but at this point we have not implemented a mechanism to attribute them a contribution measure.

We ignore the time difference between edits. When pages receive many views with little editing, it suggests that the article is substantially correct; perhaps later edits are due to changing facts, and not because of poor quality. Articles which are the subject of current events are particularly likely to have their edit quality misjudged. Relatedly, grouping revisions by author ignores the fact that edits separated by days or months are less related and have most likely been reviewed by others.

Comparing Contributions

Defining multiple contribution measures affords us the opportunity to examine and quantify the user behaviors over the large scale of edits performed. We looked at the list of all blocked authors. [2] We separated them from the others with the objective of determining how many of these authors met our definition of vandals. We were surprised to note that over 51% of authors had $q^{10}_{tdecay} > 0.95$ and 39% of authors had $q^{10}_{elong} > 0.9$. In fact, over 47% of the blocked authors make text contributions that have an average text quality over 0.95. Similarly, over 32% of the these authors make edit contributions that have an average edit quality over 0.9. We note that 11.2% of these authors qualify as vandals by our measure, based on their average edit quality and 24.9% qualify as vandals based on their average text quality. But a large percentage of the authors in the blocked authors list are not vandals, as determined by our definition. For example, authors 3362 and 10784 are both blocked, but are over the $99th$ percentile on EditLongevity, TextLongevity and TextLongevityWithPenalty.

The highest ranks across all contributions were secured by authors 3903 and *AntiVandalBot*. Author 3903 had the top rank with respect to measures TextOnly, Text-

[2] Retrieved on May 8, 2008, directly from the Wikipedia database. It corresponds to the data available at `http://en.wikipedia.org/wiki/Wikipedia:List_of_banned_users`.

Longevity, TextLongevityWithPenalty and TenRevisions. *AntiVandalBot* had the top rank with respect to the measures EditLongevity and EditOnly. Interestingly, *SmackBot* was the second highest scorer after author 3903 on measures TextLongevity and TextLongevityWithPenalty.

5.7 Conclusions

As group collaboration becomes more prevalent, the problem of how to compute author contributions becomes increasingly relevant. Our motivation was to explore simple models of user behavior that can be incorporated into reputation systems (e.g., [2]), but we feel that factoring in a notion of *quality* alongside *quantity* can also be revealing in studies about user behavior and the amount of useful information added to the Wikipedia because it cancels out the work of vandals and the work of those who fix the vandalism. We have presented and compared several possible ways to measure author contribution, including two measures popularized by previous works. What we discovered is that there is substantial agreement between the measures for clear cases of valuable contributions, and varying results for authors making questionable contributions.

There are several measures we have defined that have a desirable property, namely, giving credit where it is due and making sure that authors who make short-lived contributions get a low score. We believe that TextLongevity or EditLongevity are equally viable as contribution measures, depending on what behavior should be encouraged in users.

The EditLongevity measure is a very interesting measure in our opinion. This measure uses edit distance (as counted in words) to measure the size of the contribution while taking into account the longevity of that contribution, quantified using the

edit quality measure q^{10}_{elong}. Since the edit quality measures how much an edit takes a page towards a future version of that page, we find this a good way of measuring contribution. The TextLongevityWithPenalty measure is good at identifying vandals, but fails as a good contribution measure as it does not always reward good edits (such as those authors who revert vandalism).

As a side effect of our analysis and comparison, we were able to identify some unusual author behaviors. We discovered that the highest contributor by our edit measures was a bot, the second highest contributor by TextLongevity and TextLongevityWithPenalty was again a bot, and that there are evil bots which create a significant amount of vandalism. We also discovered that making large and good text and edit contributions are not always sufficient to be in good standing on the Wikipedia.

There are several directions for future work on measuring author contributions. Our approach has been to consider *content-driven* quality measures, where no human judgements are necessary, focusing on various measures of *longevity*. Other quality measures are equally viable, such as a "thumbs-up or thumbs-down" rating system for contributions, and the challenge is in both defining them and interpreting the results within context. For example, we have described long-lived content as "good," but might have also described the content as having reached a group consensus. Factoring quality measures into contribution measures can be useful in other collaborative endeavors such as source code archives, or even forum postings. Again, interpretation should be approached with care; for example, a wiki on current events might value short-lived content. Finally, although we have observed that there is general agreement between the measures we have examined, the differences between them highlight groups of users who behave unusually. We have tried to explain a few of the prominent groups, but there is still much to understand about various behaviors that users exhibit.

Chapter 6

Reputation

6.1 Introduction

An obvious way to build a reputation system would be to build one modeled on eBay or Amazon's rating system: elicit votes from users on the quality of a revision or the work of an author. We had several concerns about this style of system: lack of adoption, Sybil attacks through puppet votes, block voting by cliques of users, and disrupting the current user experience of the Wikipedia community. This led us to the idea of *implicit voting* through actions already taken by the community as part of their normal activities, an idea related to *revealed preferences* [88]. For example, reading an article and not making an edit to it is an implicit vote (though perhaps not a strong one) that the article is of good quality; the more page views an article has, the more we can believe that there are no errors in it. We call reputation systems that do not use explicit input from users, but instead depend on the actual content, *content-driven*.

This chapter reprints material originally published as [2].

The simplest idea for a content-driven reputation system would measure how much text an author contributed. During the course of our research, however, we realized that there are two distinct ways that authors contribute to the Wikipedia: by adding new content, and by revising existing content. Both are important to consider, since several users will adopt one contribution style and not the other. In Chapter 5, we propose several different ways to measure the contribution of authors. Two of those models, TenRevisions and EditLongevity, are the foundation of the reputation system analyzed in this chapter. There are several constants in our final model, which were assigned values by optimizing for the heuristic that author reputation at the time of an edit should be correlated with the edit longevity of that revision. We use the quality measures developed in Chapter 4 to evaluate the performance of our reputation system.

There are several possible applications of computing a reputation value for authors (for example, to grant or deny editing rights to crucial pages [10]); within the WikiTrust project, we use reputation to drive a trust system for Wikipedia content [1].[1] The fact that authors can only comment on other authors by making contributions themselves discourages users from attacking each other in an unproductive way, because they risk their own reputation in the process.

6.2 Related Work

The work most closely related to this one is [127], where the revision history of a Wikipedia article is used to compute a trust value for the article. Dynamic Bayesian networks are used to model the evolution of trust level over the revisions. At each edit, the inputs to the network are a priori models of trust of authors (determined by their

[1]This trust coloring is publicly available through our Firefox plugin: `https://addons.mozilla.org/en-US/firefox/addon/wikitrust/`

Wikipedia ranks), and the amount of added and deleted text. The paper shows that this approach can be used to predict the quality of an article; for instance, it can be used to predict when an article in a test set can be used as a featured article. In that work, author trustworthiness is taken as input; we compute author reputation as output. Several approaches for computing text trust are outlined in [64]. A simpler approach to text trust, based solely on text age, is advocated in [22].

Reputation systems in e-commerce and social networks has been extensively studied [86, 27, 51, 33]; the reputation in those systems is generally user-driven, rather than content-driven as in our case. Related is also work on trust in social networks [40, 37], as well as search ranking for web pages [55, 77]. Vandalism detection is a closely related problem; we view reputation as an input to a vandalism detection system and discuss this application in Chapter 7.

Our work is a form of analysis on the evolution of text over time; other research has also investigated such evolution. The history flow of text contributed by Wikipedia authors has been studied with flow visualization methods in [107]; the results have been used to analyze a number of interesting patterns in the content evolution of Wikipedia articles. Work on mining software revision logs [60] is similar in its emphasis of in-depth analysis of revision logs; the aim there, however, is to find revision patterns and indicators that point to software defects, rather than to develop a notion of author reputation.

6.3 A Content-Driven Reputation System

We propose a *content-driven* reputation system in order to preserve the current user experience of the Wikipedia. There are two behaviors that we choose to promote as desirable: contributing text to articles, and editing text. Adding text to articles is a

necessary behavior for the Wikipedia to acquire new knowledge and continue to expand existing articles. However, adding text is not sufficient to make the Wikipedia a useful resource — the text must be edited for formatting and readability, and vandalism must be removed, so the removal and rearrangement of text is also very important.

To measure text contributions and the amount of editing performed by an author, we examine only the revision history of each article and use the text differencing algorithm of Chapter 3 to assist in computing these values. Deriving implicit judgements from the history was an important challenge in this work, and our reputation system builds upon the ideas developed in Chapter 4 for judgements. Thus, the central premise of our analysis is the notion that later authors of an article are implicitly judging the work of earlier authors. In order to reduce the negative impact of vandals, we scale the computed judgement by the reputation of the judge, so that high-reputation judges have a larger influence on improving the reputation of the author being judged. This limits the damage that vandals can cause by creating multiple anonymous accounts.

Text Contributions

We deem a text contribution to be *useful* to the Wikipedia if it *survives* over multiple revisions. That is, if later editors choose to preserve the text, then implicitly they are voting that the contribution was of good quality. We would like to increase the reputation of the author making the text contribution by an amount relative to the size of their contribution (thus encouraging larger contributions), but also factoring in the reputation of the judge so as to reduce the weight of low-reputation users (who might be vandals).

Formally, for an article, $a \in \mathbb{A}$, we consider two versions, $v_i, v_j \in \mathbb{V}[a]$, where $i < j$ so that $A_i = \mathrm{RevAuthor}(v_i)$ is the author we are adjusting the reputation of, and $A_j = \mathrm{RevAuthor}(v_j)$ is the author making the implicit judgement, where we choose

j such that $A_i \neq A_j$.[2] The amount of text contributed in v_i that survives to v_j is then given by $TSurv_a(i,j)$, as defined in Equation 4.1. Thus, we propose the following rule:

Rule 1. **(reputation update due to text survival)**

We update the reputation of $\mathrm{RevAuthor}(v_i)$ by considering each of the ten following revisions as judges,[3] $v_j \in J_a^{10}(i)$, and update the reputation by the amount:

$$c_{scale} \cdot c_{text} \cdot \frac{TSurv_a(i,j)}{TSurv_a(i,i)} \cdot (TSurv_a(i,i))^{c_{len}} \cdot \log(1 + R(v_j)),$$

where $j = \mathrm{RevPos}(v_j)$ is the version position of v_j in $\mathbb{V}[a]$ (and consequently, $0 < j - i \leq 10$), $c_{scale} > 0$, $c_{text} \in [0,1]$, and $c_{len} \in [0,1]$ are parameters, and where $R(v_j)$ is the reputation of $\mathrm{RevAuthor}(v_j)$ at the time v_j is performed.

In this rule, $TSurv_a(i,j)/TSurv_a(i,i)$ is the fraction of text introduced at version v_i that is still present in version v_j; this is a measure of the "quality" of v_i. The quantity $\log(1 + R(v_j))$ is the "weight" of the reputation of $\mathrm{RevAuthor}(v_j)$; that is, how much the reputation of $\mathrm{RevAuthor}(v_j)$ lends credibility to the judgements made by $\mathrm{RevAuthor}(v_j)$. In Chapter 5, we saw that for any measure we investigated, only a few regular contributors dominate the majority of users by several orders of magnitude. We therefore use a logarithmic weight for reputation to ensure that the feedback coming

[2]Recall from the definition of $\mathbb{V}[a]$ in Chapter 2 that the revisions are *filtered* so that there are no consecutive revisions by the same author. Where the same author edits multiple revisions in a row, only the most recent is kept. This has the effect of collapsing multiple checkpoint revisions into a single edit.

[3]See Definition 4.6.

from new authors is not completely overridden by the feedback coming from the dominant contributors. The parameters c_{scale}, c_{text} and c_{len} were determined experimentally via an optimization process, described in Section 6.4. The parameter $c_{len} \in [0, 1]$ is an exponent that specifies how to take into account the length of the original contribution: if $c_{len} = 1$, then the increment is proportional to the length of the original contribution; if $c_{len} = 0$, then the increment does not depend on the length of the original contribution. The parameter c_{scale} specifies how much the reputation should vary in response to individual feedback. The parameter c_{text} specifies how much the feedback should depends on residual text (Rule 1) or residual edit (Rule 2, presented later).

To give feedback on a revision, the rule considers at most 10 successive versions. This ensures that contributors to early versions of articles do not accumulate disproportionate amounts of reputation. We considered basing the limit on time, rather than on the number of versions, but each Wikipedia article has its own rate of change: using the number of versions ensures that fast and slow-changing pages are treated in a similar fashion.

Edit Contributions

Similar to text contributions, we define edit contributions to be *useful* if they *survive* revision by multiple later authors. We use the notion of *edit longevity* defined in Equation 4.4 as a guide to the quality judgement made by $\mathrm{RevAuthor}(v_j)$ of the work by $\mathrm{RevAuthor}(v_i)$:

$$ELong_a(i, i-1, j) = \frac{d_a(i-1, j) - d_a(i, j)}{d_a(i-1, i)}$$

Intuitively, this formula computes whether the work in going from version v_{i-1} to version v_i brings the article closer to how the article will look in the future (as seen from the point of view of version v_j). If the edit distance, $d()$, satisfies the triangular inequality, then $ELong_a(i, i-1, j) \in [-1, 1]$; for many choices of $d()$, the triangle inequality is not satisfied, so we restrict the value of $ELong()$ to be in the range $[-1, 1]$ by fixing the value to the closest endpoint when it falls outside of the range. For two consecutive edits v_i, v_{i+1}, if v_i is completely undone in v_{i+1} (as is common when v_i introduces spam or is some other kind of vandalism), then $ELong_a(i, i-1, i+1) = -1$; if v_{i+1} completely preserves the work of v_i, then $ELong_a(i, i-1, i+1) = +1$. Values in between the two extremes represent how much of the edit is preserved or undone through revisions up to and including v_j.

Note that $ELong_a(i, i-1, j) < 0$ only when $d_a(i-1, j) < d_a(i, j)$, that is, when v_j is closer to the version v_{i-1} (the *preceding* version), than to version v_i. In other words, $\mathrm{RevAuthor}(v_j)$ votes to lower the reputation of $\mathrm{RevAuthor}(v_i)$ only when the preceding v_{i-1} is more like v_j than v_i is. We use the following rule for updating reputations based on edit contributions.

Rule 2. **(reputation update due to edit survival)**

We update the reputation of $\mathrm{RevAuthor}(v_i)$ by using the three following revisions as judges, $v_j \in J_a^3(i)$, to compute the following value:

$$q = \frac{c_{slack} \cdot d_a(i-1, j) - d_a(i, j)}{d_a(i-1, i)}$$

where $j = \mathrm{RevPos}(v_j)$. Since q is undefined when $d_a(i-1, i) = 0$, we take that to be a special case where no reputation should accrue to the author

of version v_i (i.e., we set $q = 0$ in that case). We also define a *punishing* function

$$p(q) = \begin{cases} 1 & \text{if } q \geq 0, \\ c_{punish} & \text{if } q < 0. \end{cases}$$

The reputation of $\mathrm{RevAuthor}(v_i)$ is then increased according to the following formula:

$$q \cdot p(q) \cdot c_{scale} \cdot (1 - c_{text}) \cdot (d_a(i-1, i))^{c_{len}} \cdot \log(1 + R(v_j))$$

In this rule, $c_{punish} \geq 1$, $c_{slack} \geq 1$, $c_{scale} > 0$, $c_{text} \in [0, 1]$, and $c_{len} \in [0, 1]$ are parameters, and $R(v_j)$ is the reputation of $\mathrm{RevAuthor}(v_j)$ at the time version v_j is created.

We constructed this rule to use a modified form of Equation 4.4; the parameter c_{slack}, when it is greater than one, is used to spare a_i from punishment when the revision from v_{i-1} to v_i is only slightly counterproductive. On the other hand, when punishment is incurred, its magnitude is magnified by the amount c_{punish}, raising the reputation cost of edits that are later undone. We see amplifying the punishment as being instrumental to making the threat a credible one. Without amplification, a rogue contributor could use the reputation gained in one part of the Wikipedia to constantly destroy a small set of articles elsewhere. Amplification makes this harder to achieve.

The parameters c_{slack} and c_{punish}, as well as c_{scale}, c_{text} and c_{len}, were determined via an optimization process described in Section 6.4.

Computing Content-Driven Reputation

We compute the reputation for Wikipedia authors as follows. We examine all revisions in chronological order, thus simulating the same order in which they were submitted to the Wikipedia servers. We initialize the reputations of all authors to the value 0.1; the reputation of anonymous authors is fixed to 0.1. We choose a positive initial value to ensure that the weight, $\log(1+r)$, of an initial reputation, $r = 0.1$, is non-zero, priming the process of reputation computation. This choice of initial value is not particularly critical (the parameter c_{scale} may need to be adjusted for optimal performance, if this initial value is changed). As the revisions are processed,we use Rules 1 and 2 to determine which authors are being judged by the revision being process, and update the reputations of those authors accordingly. When updating reputations, we ensure that they never become negative, and that they never grow beyond a bound $c_{maxrep} > 0$. The constant c_{maxrep} prevents frequent contributors from accumulating unbounded amounts of reputation, and becoming essentially immune to negative feedback. The value of c_{maxrep} was also determined via optimization techniques, as described in Section 6.4.

Wikipedia allows users to register and create an *author* identity whenever (and as often as) they wish. As a consequence, we need to make the initial reputation of new authors very low, close to the minimum possible (in our case, 0). If we made the initial reputation of new authors any higher, then authors, after committing revisions that damage their reputation, would simply re-register as new users to gain the higher value. An unfortunate side-effect of allowing people to obtain new identities at will is that we cannot presume that people are innocent until proven otherwise: we have to assign to newcomers the same reputation as proven offenders.[4] This is a contributing

[4]Perhaps a way out of this conundrum is to use the methods of vandalism detectors (discussed in Section 7.2) to determine an initial reputation based on other factors about the edit.

factor to our reputation having low precision; many authors who have low reputation still perform very good quality revisions, as they are simply new authors rather than proven offenders.

6.4 Evaluation Metrics

In developing a reputation system, one must ask "what is it intended to signal?" For WikiTrust, our hope was that a high reputation would signal that edits made by the author were likely to be of good quality, while a low reputation would signal that the edit was of poor or unknown quality. Evaluation of the system becomes the crucial factor, so that users can compare one system to another.

We evaluate our reputation system by using the quality measures of Chapter 4 to define two binary classifications, and then calculate our reputation system's precision and recall for correctly classifying each revision according to those classifications. We observe that both text longevity and edit longevity are computed based on the evolution of the article text *after* the time that revision v_i is created, while $R(v_i)$ is computed based on events *before* the time of v_i, so that a comparison between them isn't predisposed to showing a correlation.

To formally define this framework, we take the view that revisions are generated by a probabilistic process, with $\mathbb{V}$ as the list of outcomes from that process. We associate with each revision a probability mass (a weight) proportional to the number of words affected by the edit. This compensates for our setting of *filtered* revisions, where we combine consecutive revisions made by the same author; in such a setting, the unit of a "revision" is somewhat arbitrary, while weighting scales with the net amount of work done by each author. Each of our two quality measures has a weighting that is appropriate to it; given $v_i \in \mathbb{V}[a]$, for some article $a \in \mathbb{A}$, where $i = \mathrm{RevPos}(v_i)$ as

usual, we define the probability mass to scale a revision by as:

$$\rho_e(v_i) = d_a(i-1, i), \qquad \text{for edit longevity.}$$
$$\rho_t(v_i) = TSurv_a(i, i), \qquad \text{for text longevity.}$$

We define our categories by choosing a partition for each measure, and define three random variables $S_e, S_t, L : \mathbb{V} \mapsto \{0, 1\}$ as follows:

- We say that the new text added in version v_i is *short-lived text* if $q^{10}_{tdecay}(i)$ is at the low end of the range. We define $S_t(v_i) = 1$ if $q^{10}_{tdecay}(i) \leq 0.2$, and $S_t(v_i) = 0$ otherwise.[5] This indicates that at most 20% of new text, on average, survives from one version to the next.

- We say that the edit performed in taking version v_{i-1} to version v_i is a *short-lived edit* if $q^3_{elong}(i)$ is low. Specifically, $S_e(v_i) = 1$ if $q^3_{elong}(i) \leq -0.8$, and $S_e(v_i) = 0$ otherwise.[6]

- We also partition revisions according to whether they are *low-reputation* or not. We define low-reputation similarly to the quality measures, as $L(v_i) = 1$ if $\log(1 + R(v_i)) \leq \log(1 + c_{maxrep})/5$; $L(v_i) = 0$ otherwise, and again we have chosen a partition that represents the lowest 20% of the range after logarithmic scaling. Note that the reputation of $\mathrm{RevAuthor}(v_i)$ does not actually change at the time of version v_i's creation; the reputation of the author of a revision is adjusted as judges become available.

[5] Recall that $q^{10}_{tdecay}(i)$ is given by the solution to Equation 4.2.

[6] The quality, $q^3_{elong}(i)$, is given by Definition 4.7.

The precision $\textit{prec}_t$ and recall $\textit{rec}_t$ for short-lived text, and the precision $\textit{prec}_e$ and recall $\textit{rec}_e$ for short-lived edits, are defined as:

$$\textit{prec}_t = \Pr(S_t{=}1 \mid L{=}1) \qquad \textit{rec}_t = \Pr(L{=}1 \mid S_t{=}1)$$
$$\textit{prec}_e = \Pr(S_e{=}1 \mid L{=}1) \qquad \textit{rec}_e = \Pr(L{=}1 \mid S_e{=}1).$$

These quantities can be computed as usual; for instance,

$$\Pr(S_e = 1 \mid L = 1) = \frac{\sum_{v \in \mathbb{V}} S_e(v) \cdot L(v) \cdot \rho_e(v)}{\sum_{v \in \mathbb{V}} L(v) \cdot \rho_e(v)}.$$

We also define the *boost* that knowing reputation gives to predicting a short-lived revision:

$$\textit{boost}_e = \frac{\Pr(S_e = 1 \mid L = 1)}{\Pr(S_e = 1)} = \frac{\Pr(S_e = 1, L = 1)}{\Pr(S_e = 1) \cdot \Pr(L = 1)}$$
$$\textit{boost}_t = \frac{\Pr(S_t = 1 \mid L = 1)}{\Pr(S_t = 1)} = \frac{\Pr(S_t = 1, L = 1)}{\Pr(S_t = 1) \cdot \Pr(L = 1)}$$

Intuitively, $\textit{boost}_e$ indicates how much more likely than average it is that edits produced by low-reputation authors are short-lived. The quantity $\textit{boost}_t$ has a similar meaning.

Our last indicators of quality are the *coefficients of constraint* [19, 21]:

$$\kappa_e = I_e(S_e, L)/H_e(L) \qquad \kappa_t = I_t(S_t, L)/H_t(L),$$

where I_e is the *mutual information* of S_e and L, and H_e is the entropy of L; similarly for $I_t(S_t, L)$ and $H_t(L)$. The quantity κ_e is the fraction of the entropy of the edit longevity which can be explained by the reputation of the author; this is an information-theoretic measure of correlation. The quantity κ_t has an analogous meaning.

To assign a value to the coefficients c_{scale}, c_{slack}, c_{punish}, c_{text}, c_{len}, and c_{maxrep}, we implemented a search procedure, whose goal was to find values for the parameters that maximized a given objective function. We applied the search procedure to the Italian Wikipedia, reserving the French Wikipedia for validation once the coefficients were determined. We experimented with κ_e and $prec_e \cdot rec_e$ as objective functions, and they gave very similar results.

6.5 Experimental Results

To evaluate our content-driven reputation, we considered two Wikipedias:

- The Italian Wikipedia, consisting of 154,621 articles and 714,280 *filtered* revisions; we used a snapshot dated December 11, 2005.
- The French Wikipedia, consisting of 536,930 articles and 4,837,243 *filtered* revisions; we used a snapshot dated October 14, 2006.

In both Wikipedias, we studied only `NS_MAIN` pages, which correspond to ordinary articles (other pages are used as comment pages, or have other specialized purposes). Moreover, to allow the accurate computation of our quality measures which require multiple judges, we used only revisions that occurred before October 31, 2005 for the Italian Wikipedia, and before July 31, 2006 for the French one. Our algorithms for computing content-driven reputation depend on the value of six parameters, as mentioned earlier. We determined values for these parameters by searching the parameter space to optimize the coefficient of constraint κ_e, using the Italian Wikipedia as a training set;

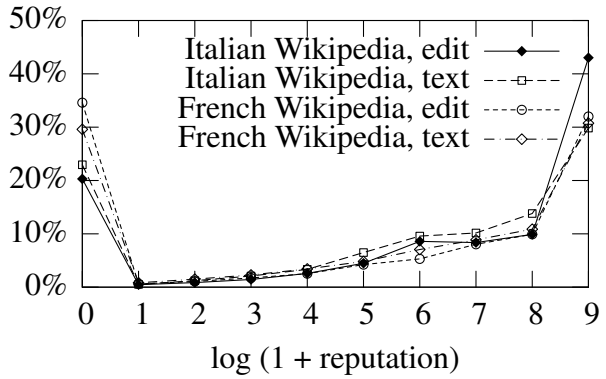


Figure 6.1: Percentage of text and edit contributed to the Italian and French Wikipedias, according to author reputation. The data includes anonymous authors.

the values we determined are:

$$c_{scale} = 13.08 \qquad c_{punish} = 19.09 \qquad c_{len} = 0.60$$

$$c_{slack} = 2.20 \qquad c_{text} = 0.60 \qquad c_{maxrep} = 22026$$

Figure 6.1 provides a breakdown of the amount of edits and text additions performed, according to the reputation of the author. We find an essentially bimodal distribution: the overwhelming majority of revisions are authored by the lowest and highest reputation users in the system. Note the similarity to the results of Chapter 5, especially Figures 5.1 and 5.3; we again see a bimodal distribution with an immediate drop-off beyond the lowest reputation, and an upwards trend for frequency as we move towards the higher end of the reputation spectrum. We believe that this might reflect the bimodal behavior of the community; many users tend to make only a small number of edits, and a few users contribute a great deal (see Figure 5.4).

Precision and Recall

We analyzed the Italian and French Wikipedias using the parameter values discovered by our optimization procedure. The results are summarized in Table 6.1. The results are better for the larger French Wikipedia; in particular, our reputation's ability to predict short-lived edits is better on the French than on the Italian Wikipedias. We are not sure whether this depends on different dynamics in the two Wikipedias, or whether it is due to the greater age (and size) of the French Wikipedia. We see that edits performed by low-reputation authors are four times as likely as the average to be short-lived.

Manual Annotation

To investigate how many of the edits had a short life due to bad quality, we asked a group of seven volunteers to rate revisions made to the Italian Wikipedia. We selected the revisions to be ranked so that they contained representatives of all 4 combinations of high/low reputation author, and high/low longevity. We asked the volunteers to rate the revisions with $+1$ (good), 0 (neutral), and -1 (bad); in total, 680 revisions were ranked. The results, summarized in Table 6.2, are striking. Of the short-lived edits performed by low-reputation users, fully 66% were judged bad. On the other hand, less than 19% of the short-lived edits performed by high-reputation users were judged bad. We analyzed in detail the relationship between user reputation, and the percentage of short-lived text and edits that users considered bad. Using these results, we computed the approximate recall factors on the Italian Wikipedia of content-driven reputation for *bad* edits, as judged by users, rather than short-lived ones:

- The recall for short-lived edits that are judged to be bad is over 49%.
- The recall for short-lived text that is judged to be bad is over 79%.

	Precision		Recall		Boost		Coeff. of constr.	
	Edit $prec_e$	Text $prec_t$	Edit rec_e	Text rec_t	Edit $boost_e$	Text $boost_t$	Edit κ_e	Text κ_t
Excluding anonymous authors:								
French Wikipedia	23.92%	5.85%	32.24%	37.80%	4.21×	4.51×	7.33	6.29
Italian Wikipedia	14.15%	3.94%	19.39%	38.69%	4.03×	5.83×	3.35	7.17
Including anonymous authors:								
French Wikipedia	48.94%	19.01%	82.86%	90.42%	2.35×	2.97×	25.29	23.00
Italian Wikipedia	30.57%	7.64%	71.29%	84.09%	3.43×	3.58×	19.83	17.49

Table 6.1: Summary of the performance of content-driven reputation for the Italian and French Wikipedias.

These results clearly indicate that our content-driven reputation is a very effective tool for spotting, at the moment they are introduced, bad contributions that will later be undone. There is some margin of error in this data, as our basis for evaluation is a small number of manually-rated revisions, and human judgement on the same revisions often contained discrepancies.

The fact that so few of the short-lived edits performed by high-reputation authors were judged to be of bad quality points to the fact that edits can be undone for reasons unrelated to quality. Many Wikipedia articles deal with current events; edits to those articles are undone regularly, even though they may be of good quality. Our algorithms do not treat in any special way current-events pages. Other Wikipedia edits are administrative in nature, flagging pages that need work or formatting; when these flags are removed, we classify it as text deletion. Furthermore, our algorithms do not track text across articles, so that when text is moved from one article to another, it is classified as deleted from the source article.

From Table 6.1, we note that the precision is low, by search standards. Our problem, however, is a prediction problem, not a retrieval problem, and thus it is intrinsically different. The group of authors with low reputation includes many authors who are good contributors, but who are new to the Wikipedia, so that they have not had time yet to build up their reputation.

Comparison with Edit-Count Reputation

We compared the performance of our content-driven reputation to another basic form of reputation: edit count. It is commonly believed that, as Wikipedia authors gain experience (through revision comments, talk pages, and reading articles on Wikipedia

Reputation	Judged bad	Judged good
Short-lived edits:		
Low [0.0–0.2]	66 %	19 %
Normal [0.2–1.0]	16 %	68 %
Short-lived text:		
Low [0.0–0.2]	74 %	13 %
Normal [0.2–1.0]	14 %	85 %

Table 6.2: User ranking of short-lived edits and text, as a function of author reputation, for the Italian Wikipedia. We presented edit differences to a test group of users, and asked users to rate whether the edit was good or bad. In square brackets, we give the interval where the normalized value $\log(1+r)/\log(1+c_{maxrep})$ of a reputation r falls. The precentages do not add to 100%, because users could also rank changes as "neutral".

standards), the quality of their submissions goes up.[7] Hence, it is reasonable to take edit count, that is, the number of edits performed, as a form of reputation. We compare the performance of edit count, and of content-driven reputation, in Table 6.3. The comparison does not include anonymous authors, as we do not have a meaningful notion of edit-count for them. According to our metrics, content-driven reputation performs slightly better than edit-count reputation on both the Italian and French Wikipedias.

We believe that one reason edit-count based reputation performs well in our measurements is that authors, after performing edits that are often criticized and reverted, commonly either give up their identity in favor of a "fresh" one, thus zeroing their edit-count reputation and "punishing" themselves, or stop contributing to the Wikipedia altogether.[8] However, we believe that the good performance of edit count is an artifact, due to the fact that edit count is applied to an already-existing history of contributions. Were it announced that edit count is the chosen notion of reputation, authors would

[7] See [44] for an analysis that refutes this assumption, however.

[8] This is consistent with the conclusions presented in [44].

	Precision		Recall		Boost		Coeff. of constr.	
	Edit $prec_e$	Text $prec_t$	Edit rec_e	Text rec_t	Edit $boost_e$	Text $boost_t$	Edit κ_e	Text κ_t
Italian Wikipedia:								
Content-driven reputation	14.15	3.94	19.39	38.69	4.03	5.83	3.35	7.17
Edit count as reputation	11.50	3.32	19.09	39.52	3.27	4.91	2.53	6.35
French Wikipedia:								
Content-driven reputation	23.92	5.85	32.24	37.80	4.21	4.51	7.33	6.29
Edit count as reputation	21.62	5.63	28.30	37.92	3.81	4.34	5.61	6.08

Table 6.3: Summary of the performance of content-driven reputation over the Italian and French Wikipedias. All data are expressed as percentages. Anonymous authors are not included in the comparison. Precision is the probability that the text or edit longevity is low, given that the reputation is low. Recall is the probability that the reputation is low, given that the text or edit longevity is low.

most likely modify their behavior in a way that both rendered edit count useless, and damaged the Wikipedia. For instance, it is likely that, were edit count the measure of reputation, authors would adopt strategies (and automated robots) for performing very many unneeded edits to the Wikipedia, causing instability and damage. In other words, edit count as reputation measure has very little prescriptive value that would benefit the Wikipedia. In contrast, we believe our content-driven reputation, by prizing long-lasting edits and content, would encourage constructive behavior on the part of the authors.

Text Age and Author Reputation as Trust Criteria

The age of text in the Wikipedia is often considered an indicator of text trustworthiness, the idea being that text that has been part of an article for a longer time has been vetted by more contributors, and thus, it is more likely to be correct [22]. We were interested in testing the hypothesis that author reputation, in addition to text age, can be a useful indicator of trustworthiness, especially for text that has just been added to a page, and thus that has not yet been vetted by other contributors. Let *fresh text* be the text that has just been inserted in a Wikipedia article. We considered all text that is fresh in all the Italian Wikipedia, and we measured that 3.87 % of this fresh text is deleted in the next revision. In other words, $\Pr(\text{deleted} \mid \text{fresh}) = 0.0387$. We then repeated the measurement for text that is both fresh, and is due to a low-reputation author: 6.36 % of it was deleted in the next revision, or $\Pr(\text{deleted} \mid \text{fresh and low-reputation}) = 0.0636$. This indicates that author reputation is a useful factor in predicting the survival probability of fresh text, if not directly its trustworthiness. Indeed, as remarked above, since text can be deleted for a number of reasons aside from bad quality, author reputation is most likely a better indicator of trustworthiness than these figures indicate. We investigate a

method for computing the "reputation of text" that is based on the ideas presented here in [1].

6.6 Conclusions

In this chapter, we propose a reputation system for authors to allow us to make an educated guess at the quality of a revision when it is first made. This reputation system is build atop notions developed as quality measures in Chapter 4, which are in turn built atop a difference algorithm defined in Chapter 3.

To validate the effectiveness of the WikiTrust reputation system, we evaluate a variety of measures. We find that short-lived text and short-lived edits are correlated with low-reputation, and that manual examination of the edits by a small group of reviewers has high agreement with the assessment by our reputation system. We also compare favorably against a reputation based purely on the number of edits made by authors (the so-called *edit count* reputation [22]), but without the same exposure to simple reputation attacks such as breaking up a large edit into smaller edits.

Chapter 7

Vandalism Detection

7.1 Introduction

The Wikipedia is a shared resource of the global community, but it depends on the continuing participation of volunteers to keep it current and relevant. Anyone can edit the Wikipedia: this is both its strength and its weakness. The Wikipedia was initially a side-project of Nupedia, to facilitate collaboration on content before entering a more formal peer-review process [116]. The parent project languished in comparison to the Wikipedia because of this difference in process. Today, companies such as Facebook have recognized this strategy and incorporated "frictionless sharing" into their own services. The price paid by the Wikipedia for this increased participation is the need to guard against vandalism. Multiple studies have found that roughly 7% of edits are vandalism [79, 78]. To combat the vandalism, a group of volunteers scan the list of recent changes to catch obvious damage quickly [117].

This chapter updates the results published as [4], and includes material published in [3].

In the chapters leading to this one, we have detailed the technologies necessary to build a content-driven reputation system for authors. We first constructed a difference algorithm to compute the work done in a revision, doing so in a way which models how users think about the units of language, while maintaining performance such that the entire English Wikipedia could be evaluated in a tractable amount of time. We then proposed two methods for evaluating the quality of the work done by the users in their revisioning: text longevity and edit longevity. Finally, we use these quality measures as the basis for rules in a reputation system. The output of this reputation system is an estimate of the balance of past positive contributions over past negative contributions, which we evaluate as a predictor of the future quality of revisions by the same author.

As part of the PAN 2010 Workshop on vandalism detection[1], a competition was organized to test vandalism detection systems with a single evaluation measure. Our research group submitted a system based on features derived from WikiTrust [4], leaving out the actual reputation scores due to our lack of historical reputation values for authors. In this chapter, we revisit that work and update it by including the reputation score of authors at the time of their edits.

7.2 Related Work

Wikipedia's official statement of vandalism defines it as "a *deliberate* attempt to compromise the integrity of Wikipedia."[2] It is, of course, impossible to know the motivations of individuals, so this definition relies on human intelligence to determine vandalism on a case-by-case basis — that is, "I know it when I see it,"[3] but there is no precise

[1] `http://www.webis.de/research/events/pan-10`, Task 2

[2] `http://en.wikipedia.org/wiki/Wikipedia:Vandalism`

[3] Justice Potter Stewart in Jacobellis v. Ohio, 378 U.S. 184 (1964)

definition. Some researchers have undertaken the task of more formally defining a taxonomy of vandalism [107, 81, 18], but nearly all research on vandalism detection uses one of a small number of (convenient) definitions for purposes of obtaining an annotated corpus: **manual annotation** uses human intelligence to infer the intentions of the editor [79, 18, 112, 78], **reverts** are notations by the community when it feels that vandalism has taken place [95, 48, 8], **rollbacks** are disapprovals by Wikipedia Administrators [112], and **edit quality** generalizes the idea of measuring the sentiment of the community [2, 29]. There is an obvious variation from *manual* to *automatic* annotation in these choices, but there is another difference between them: external judgement from outside the Wikipedia community, internal explicit judgement from within the community, and internal implicit judgement based on actions by the community. Ultimately, it is the community itself which decides what is vandalism (e.g., observe the stark contrast between the communities of Slashdot[4] and Hacker News[5]), and this community standard is likely to change over time (often described as the "signal-to-noise" ratio of the community; examples of changing communities include USENET and Slashdot). This argues strongly in favor of automated methods for measuring the reaction of the community, and highlights the idea that vandalism detection is a specialized form of trying to measure the "noise" in a community.

The earliest attempts at vandalism detection within the Wikipedia come directly from the user community, and try to encode a human intuition of vandalism detection into an expert system (some examples include [118, 120, 119, 15]). The largest disadvantage to this class of solutions is that building an expert system requires extensive human labor to produce the manual annotation and analysis required to derive custom

[4] `http://slashdot.org`

[5] `http://news.ycombinator.com`

rules. Primarily, the rules developed are based on features of the actual content of the edit rather than on metadata (e.g., an edit containing profanity is indicative of vandalism).

The idea that the content reveals the intent of the author is a natural one, and has been investigated by several different research groups (e.g., [79, 95, 29, 48, 18]). Casting the problem as a binary classification problem to be solved by machine learning, Potthast et al. [79] manually identify and inspect 301 incidents of vandalism to generate a feature set based on metadata and content, and build a classifier using logistic regression. Smets et al. [95] applies the "naive bayes" machine learning technique to a bag-of-words model of the edit text. Chin et al. [18] delve deeper into the field of natural language processing by constructing statistical language models of an article from its revision history. (On the topic of manual annotation, they also describe how supervised active learning can help the training process by requesting annotations for examples which will make a significant difference to the algorithm.)

A different way of looking at the content approach is the realization that appropriate content somehow "belongs together," and one way to measure that is through compression of the successive revisions of an article [95, 48]. If inappropriate content is added to the article, then the compression level is lower than it would be for text which is similar to text already in the article. This is much more powerful than the bag-of-words model, because phrases are significant and lead to better compression; nonsensical sentences that include some key words will not compress as well. A significant drawback of these compression techniques is that they require manipulation of the content of a large number of revisions from the article being edited.

Content-based analysis has the burden of having to inspect potentially large edits, but the alternative is to depend on the paucity of information available in the metadata

— many previous works have some small dependence on metadata features [79, 29, 8], but only as far as it encoded some aspect of human intuition about vandalism. Drawing inspiration from other areas of research, West et al. [112] published good results based entirely on metadata (some of which is processed into *reputations*) that indicate there is more relatedness between vandals than is readily apparent to the human eye. One particularly interesting result was that using IP geolocation to cluster users led to better predictions.

A systematic review and organization of features appears by Potthast et al. [80] as part of the competition associated with the PAN 2010 Workshop on vandalism detection. Belani [8] includes several metrics for evaluating predictors, and Potthast et al. take up the discussion with a thorough comparison of nine competitors using both the area under the precision-recall curve and the area under the receiver operating characteristic curve. Potthast et al. conclude their analysis by building a meta-classifier based on the nine entries and discover that the result performs significantly better than any single entry.

User reputation systems [127, 64, 2] have been proposed as an underlying technology for vandalism prevention or detection, and the second place entry in the PAN 2010 competition was a system based on the WikiTrust project [4]. In that entry, the WikiTrust user reputation system was not directly used due to not having a historical record of the reputation values. The work presented in this chapter updates the results of [4] by tracking the historical user reputation values and using that as an additional feature to the machine learning algorithm.

The winner of the PAN 2010 competition, by a notable margin, was an entry by Mola-Velasco [66] that extended the features originally proposed by Potthast et al. [79]. This entry was composed of 21 features (the largest in the competition) that compre-

hensively model the content of the edit, including features that rated use of language, formatting of text, compressibility with earlier text, spelling, and the size of the edit.

A follow-up work to the PAN 2010 competition explores the complementary nature of the features used by Mola-Velasco [66], WikiTrust [4], and West [112]. That work improves on earlier results, and categorizes features according to the difficulty of analysis [3].

7.3 Experiment

To test our user reputation system's effectiveness, we evaluate its performance as part of a vandalism detection system. The PAN-WVC-10 is a corpus of 32,439 edits manually annotated by at least three people as part of an Amazon Mechanical Turk task [78]. Of these edits, 2,394 (7.97%) were classified as vandalism.

Our goal is to incorporate the WikiTrust user reputations in building a model to predict vandalized edits in the PAN-WVC-10 corpus. We modified the WikiTrust code base to output the reputation score of each author over time, creating a chronology of reputation scores. This revised code was used to process the English Wikipedia dump of 30-Jan-2010, which includes the time period of edits from the PAN-WVC-10 corpus.

Using the timestamp of each edit in the PAN-WVC-10 corpus, we locate its position in the chronology and then work backwards to find the most recent reputation score of the author that appears *before* the edit was made. These reputation scores are then merged with the collection of features used by the WikiTrust submission in the PAN 2010 competition [4], and used to build a model to predict whether an edit is vandalism. The analysis we present here considers only *zero-delay* vandalism detection; that is, detecting vandalism using only features which are available at the moment an edit is made.

Classifier and Features

Our vandalism detection tool uses the open source machine learning package Weka [45] to build and evaluate a prediction model from our features. In the original PAN 2010 competition, we used an alternating decision tree (ADTree) classifier [4] because it performed well and generates models that are easy to interpret. As part of our collaboration in combining multiple vandalism detection systems [3], we evaluated the performance of the WikiTrust features[6] using a random forest classifier and discovered that the performance was increased. Anonymous users have no reputation in the WikiTrust system, and receive a reputation score of zero within the feature set.

As in [3], we use the random forest algorithm (set to create 500 trees) to build our prediction model. We select a set of features based on the information that is readily available within the pre-existing WikiTrust system, and only those which are available at the instant an edit is made. The purpose of our experiment is to answer the question, "does the WikiTrust reputation computation provide information about whether an edit is vandalism?" In order to make that judgement, we use the same features chosen in [4] and add the WikiTrust reputation score. The total set of features we used are:

- **Author reputation [Reputation].** Vandalism tends to be performed predominantly by anonymous or novice users, both of which have reputation zero in the system. This is the only feature which is new, compared to the experiment conducted in [4].

- **Author is anonymous [Anon].** The Wikipedia software associates either a username or an IP address with every edit. WikiTrust only tracks registered usernames, and records every other edit as an anonymous edit. Vandalism is of-

[6] In addition to the zero-delay features described in [4], we also included a feature measuring the length of the article for [3].

ten committed under the cover of anonymity, although many good edits are also made anonymously.

- **Time interval since the previous revision [Logtime_prev].** We compute the quantity $\log(1+t)$, where t is amount of time since the preceding revision of the same article.

- **Hour of day when revision was created [Hour_of_day].** We expect that the time of day at which the revision was created might have some influence on the frequency of vandalism. This did not have much influence in our previous work [4], but a more sophisticated version proved to contain much information [112].

- **Delta [Delta].** This feature measures the edit distance $d(v_i, v_{i-1})$ between the revision being examined and the previous revision in the same article.

- **Revision comment length [Comment_len].** The length of the comment attached to the revision. It seems unlikely that vandals would provide a comment, so we included it as a trivial feature to compute.

- **Previous text reputation histogram [P_prev_hist0 ... P_prev_hist9].** Whenever a revision is created, WikiTrust computes a separate reputation for each word of the article, where the reputation is an integer in the interval $0, \ldots, 9$ (see [1] for details of how we calculate text reputation). The reputation of a word indicates how much the word has been revised by other reputable authors; in particular, words that are inserted or moved by authors without reputation (including both novice and anonymous authors) are assigned a reputation of zero. When the revision is created, WikiTrust also computes a ten column histogram detailing how

many words of the revision have each of the ten possible reputation values, and stores the histogram in the database in an entry associated with the revision. We normalize the histogram (so that the columns sum to one) of the previous revision in the same article history.

- **Current text trust histogram [Hist0 ... Hist9].** The values of the text trust histogram for the current revision, without any normalization.

- **Histogram difference [L_delta_hist0 ... L_delta_hist9].** For each possible text trust value $i \in \{0, \ldots, 9\}$, we also computed the value of

$$\log(1 + |h(i) - h^{-}(i)|) \cdot \text{sign}(h(i) - h^{-}(i)), \tag{7.1}$$

where h is the text trust histogram for the current revision, and h^{-} is the text trust histogram for the previous revision.

7.4 Evaluation

The results of the PAN 2010 competition [80] provide a benchmark for vandalism detection systems. Although the receiver operating characteristic was used to judge the competition, the analysis provided by Potthast et al. suggests that the precision-recall curve provides better discriminatory power between models due to the larg class imbalance between vandalized and regular edits. We evaluate our predictions using both methods and place them into the context of other previously published results.

In Table 7.1, we present the confusion matrix as determined by the Weka package during stratified ten-fold cross validation. Predicting that an edit is a regular contribution has a precision of 95.4% and recall of 98.4%; in predicting vandalism, our model

	classified as	
actual class	Regular	Vandalism
Regular	29428	467
Vandalism	1430	957

Table 7.1: The confusion matrix for predicting edits of the PAN-WVC-10 corpus as regular edits or the work of vandals, during stratified ten-fold cross validation. Note that there is a large class imbalance in the distribution of how edits are truly classified.

is only able to achieve a precision of 67.2% and recall of 40.1%. Figure 7.1 shows the corresponding precision-recall curve of the resulting predictions. Calculating other evaluation measures for classification problems, we get the following performance for classifying edits as vandalism:

$$\text{Positive Predictive Value} = 67.2\%$$

$$\text{Sensitivity} = 40.1\%$$

$$\text{Specificity} = 98.4\%$$

$$\text{Accuracy} = 94.1\%$$

Receiver operating characteristic curves are typically used to evaluate the performance of binary classification algorithms but they give optimistic results when there is a large class imbalance [24], which we see in Table 7.2. The area under the precision-recall curve gives another perspective, which is correlated with the ROC for the results we present; the table includes values from other published results to provide a greater context for the evaluation. From this data, we can answer our motivating question: does including reputation as a feature result in better predictions? Adding revision metadata features and using the random forest algorithm as a classifier results in a significant im-

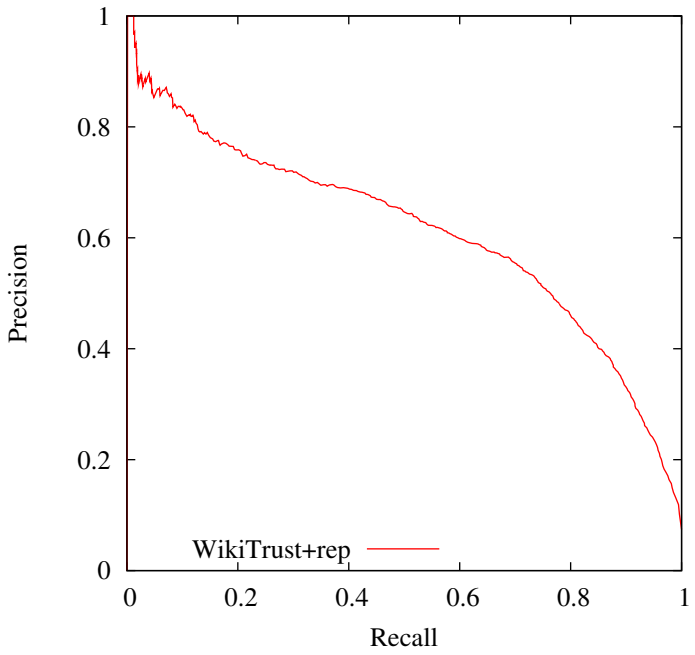


Figure 7.1: Precision-Recall curve for a vandalism detection system based on features from the WikiTrust system, including reputation. This extends the previous WikiTrust results for immediate vandalism detection [4] by including the reputation of authors at the time of each edit.

System	**AUC-PR**	**AUC-ROC**
PAN 2010 WikiTrust [80]	0.49263	0.90351
STiki (Metadata) [3]	0.52534	0.91520
PAN 2010 WikiTrust + metadata [3]	0.61047	0.93647
PAN 2010 WikiTrust + reputation	0.61152	0.94257
Mola-Velasco (NLP) [3]	0.73121	0.94567
Mola-Velasco + topic [67]	0.7541	*n/a*
PAN'10 Meta Detector [80]	0.77609	0.95689
M-V + WT + STiki [3]	0.81829	0.96902

Table 7.2: Comparison of various vandalism detection systems. The results of this chapter are labeled "PAN 2010 WikiTrust + reputation," to indicate that it builds on the results collected for and presented in [80] by including the author reputation score at the time of each edit.

provement over the original WikiTrust results [4] (shown in the third line of Table 7.2). Replacing the revision metadata features with author reputation does even better still, answering our motivating question in the affirmative.

The feature set in [4] includes a field indicating whether or not each edit is done by an anonymous user. The inclusion of reputation in our current experiment results in an increase in performance, from which we infer that the reputation system is providing information about registered users and the experience they have accrued in editing the Wikipedia.

7.5 Conclusions

Our experiment of this chapter starts with the data used by the WikiTrust submission in the PAN 2010 vandalism detection task [4], and compares the performance achieved in [80] with the performance achieved by adding author reputation as a feature. The result is a notable increase in predictive ability, demonstrating the value of maintaining author reputation scores as described in Chapter 6. The current system is limited to tracking reputations only for registered users, but we are still able to achieve a precision of 67.2% and recall of 40.1% during stratified ten-fold cross validation. Due to the large class imbalance, there's the possibility that the stratification leads to small training sets that don't cover the diversity of behaviors exhibited by vandals, limiting the predictive ability of machine learning algorithms.

Although the results of our experiment were quite positive, there are several further avenues that can be explored. West et al. [112] create reputations for several different "entities" based on edit metadata (e.g., article reputation, category reputation, and country reputation based on IP geolocation), but the entities and method of computing reputation are completely different from the choices made within WikiTrust. The idea

of using geographic entities can be applied in WikiTrust to create "author reputations" for anonymous authors, which we speculate will improve the performance further.

As shown in [3], the construction of features is complementary between WikiTrust and STiki, and performs very well when combined, but the actual formulas employed to compute reputation are ad hoc in nature. Is there some principle by which reputation algorithms can be constructed to give better results? Or a set of design choices that can be laid out [26] with known benefits and costs? How can we choose the scope of entities to construct reputations for? We leave these questions open to pursue in future works.

Chapter 8

Conclusion

8.1 Introduction

The Future is inevitable.

History abounds with examples when the time was ripe for a particular idea. Some famous instances are the development of calculus (by Leibniz and Newton), the theory of evolution (by Darwin and Wallace), and the invention of the integrated circuit (by Kilby and Noyce).

When our group began work on this research into reputation systems for the Wikipedia, it was focused on the question of how to tell "good guys" from "bad guys" on the Wikipedia. Right away, the investigation expanded to edit quality, and the question of what *is* quality in a collaborative work where there is no single guiding hand. We then realized that our work was a way to quantify the consensus of a group of people. This is a powerful view, not only because of the potential applications, but also because now we know that it is *possible* to make such a measurement. Economists long ago discovered "revealed preferences" as a way to surreptitiously measure the internal world of the mind [88, 106], and this work widens that doorway.

It was some years later that I began work at Fujitsu Labs of America, using statistics for word usage models and medical trials, when I came to a new realization. In the flood of the data deluge (which Wikipedia analysis falls smack into the middle of), statistical summaries are how we abstract that data and come to an "understanding" of it. Our reputation system is a manifestation of our rough understanding of the kind of useful Wikipedia content we aim to promote. And so I came to think that *statistics* will be the most important field in the 21^{st} century.

Most recently, our work on vandalism detection and my work with the emerging Quantified Self group[1] has led me to realize that *data mining* is the new face of statistics; simple summarization of so much data is not useful, but data mining gives us the power to pull at the different threads of individuality and cluster like with like. We go from summaries to context-dependent probabilities.

Through a chance conversation with Bayesian statistician Owen Martin, these thoughts gelled into the notion that reputation was not just some number of stars by your name — it is the probability that you will do or say something, given the context of your demographics (i.e., the background culture that helped shape you), your past history and the present moment. That is, the models built by data mining algorithms are a form of reputation constructed around the description of user behavior.[2] This work is just the first fumbling footsteps in that line of thinking.

---

[1] `http://quantifiedself.com`

[2] Reputation systems are both descriptive and prescriptive [2]. Machine learning algorithms compute a description of past behavior but don't appear to include a prescriptive component; the key to reconciling these views is in the choice of what to predict. For example, predicting whether an edit will be "long lived" naturally takes negative behaviors into account.

8.2 Summary of this Work

The creation of a collaborative repository of knowledge is one of those inevitable ideas; we have been striving towards it for all of civilization. Our history is one of ever increasing communication and collaboration in our ambition to master our environment, from books and libraries, to the telephone and the Internet. The Wikipedia seems a natural result of this history. And perhaps just as inevitably, there are people who seek to mar the collective goals of the Wikipedia for their own gain and amusement.

The goal of our research was to develop a reputation system that could assist users by flagging content added by users who did not have a good track record, making the task of understanding the short-term history of an article easier to grasp with a glance. In working towards that goal, this research makes the following contributions:

1. Two methods for determining the "quality" of a contribution. We evaluate these methods (and the underlying difference algorithms) using the PAN 2010 vandalism detection corpus, and find that both measures perform much better than a random vandalism detector. Our *text longevity* measure uses authorship information to compute the amount of contributed text and how it survives over future revisions (its "rate of decay"). Our *edit longevity* measure uses edit distances to estimate how much "work" of an edit goes to making an article more like a future instance of the article.

2. A reputation system for authors, which adjusts the reputation of an author based on quality feedback from later authors using the ideas of text and edit longevity. Our evaluation shows that content by low-reputation authors is four times as likely as the average to be short-lived, and that better precision-recall can be achieved when used as part of a vandalism detection system.

During the course of developing and studying these primary contributions, we have also had several smaller accomplishements with enabling other technologies based on our reputation data:

text reputation Our group developed a reputation system for text [1], and created a visualization tool that is available to users as a Firefox plugin.[3]

vandalism detection The work of Chapter 7 grew out of an entry into a vandalism detection competition. That effort led to a joint publication [3] presenting results that outperformed previously published results. Our features have also been incorporated into the STiki vandalism detection tool [124].

revision selection The Wikipedia Offline project published a CD-ROM of a selection of Wikipedia content, which is distributed to school children across America. Due to space constraints, the project only selects one revision for each article, but manually reviews each choice to check for vandalism or questionable changes. We developed a system based on our vandalism detection work (and coining the term "historic vandalism detection") to identify revisions that were likely to be vandalism, narrowing the list of revisions that needed manual review.

As part of this work, we also investigated text difference algorithms and outlined several design issues in how author attribution is determined in a collaborative document. With respect to text difference algorithms, we ultimately discovered that the difference algorithm did not have a very dramatic impact on the performance of the edit longevity measure (shown in Tables 4.3 to 4.7); we would have achieved substantially similar results simply using the fastest performing algorithm.

[3] `https://addons.mozilla.org/en-US/firefox/addon/wikitrust/`

Future Work

A critical part of the training received in graduate school is the ability to ask questions that creatively expand on your existing work. There are a great number of questions left to explore in the context of WikiTrust, some of which are described here.

category reputation This is probably the one idea that we hear the most from audiences. A person can be an expert in one area but not another, but some people don't recognize their lack of expertise. By keeping track of separate reputation scores for every category, the system should be more able to predict the longevity of an edit. It is worth noting that, within the Wikipedia community, there are those editors which specialize in grammar and style rather than subject areas, so that there is some extra work that must go into classifying the type of edit (e.g., [35]). Note that the STiki project [112] includes "category reputation", but it tracks a reputation for each category globally; here we mean that reputation is multidimensional and an author can have differing reputations in different categories.

reputation as probabilities This view of reputation clarifies how to best interpret the results. It also introduces the significant question of how should probabilities be adjusted as new information becomes available. Owen Martin's work on estimating bug counts provides a nice example: you are trying to build a rocketship, and you run many tests to try to uncover bugs [63]. If the test succeeds, your estimate of the number of bugs (which is a form of reputation: "what is the probability that the system will fail?") goes down. If the test fails, how do revise your estimate of the number of bugs? And once the engineers have fixed the problem, can you be sure they haven't introduced new problems?

contribution types Some users contribute text while others perform maintenance duties, which we were able to identify by comparing different metrics in Chapter 5. Categorizing contributions by these metrics can help in the awarding of barnstars, but also might be useful as additional features in a vandalism detection system.

match quality Our greedy text differencing algorithm uses a match quality function to prioritize which matches are preferable according to the criteria described in Chapter 3. The evaluation in Chapter 4 suggests that as long as length is the primary discriminant, there is not too much difference between the different quality functions. The evaluation used is one based on the resulting predictive ability of edit quality, but is there some better way to evaluate a difference algorithm whose aim is to model the human view of a text edit?

authorship We devised an algorithm for determining the "authorship" of words in revisioned documents.[4] This algorithm is a refinement of existing difference algorithms already well-known in the literature, but we additionally outline some of the design considerations for this particular application. As described in the concluding remarks of Chapter 3, there are further refinements to be considered in how to assign authorship in a collaborative work. Identifying common idioms in the language (e.g., through the use of tf-idf [49]) and common phrases within a topic area are two cases where authorship needs to be more carefully considered. The greatest challenge here is determining a suitable evaluation for comparing solutions.

[4]This work is being explored further by the German chapter of the Wikipedia: `http://de.wikipedia.org/wiki/Benutzer:NetAction/WikiTrust`

8.3 Thoughts on Reputation

What is reputation? We know that it is a value, because your reputation goes up and down. It can be good or bad.

What is less obvious is that reputation is multidimensional. Some writers have a good reputation for writing engaging essays (e.g., Malcolm Gladwell), and some have a good reputation for writing good children's fiction (e.g., J. K. Rowling), but most of us would be doubtful if Gladwell and Rowling were to swap careers. Interestingly, we also relate reputation between dimensions — for example, someone with a good reputation for civic involvement and hard work we also presume to have a high reputation in regards to his behavior at home (for example, marital fidelity).

After so many years of working with reputation, I have come to believe that internally we use reputation as something like a probability (as in reliability theory). That is, all other things being equal, reputation is the chance that you will meet some standard of performance in a situation that calls for it. The trick is that things are never "equal" from situation to situation, so that there are unknown components of a reputation that we construct from what we know of a person's background and culture. The most direct example of this is racial stereotyping applied to a stranger.

Although reputation might lead us to jump to incorrect conclusions about someone based on limited information, they also help us to filter and sort through the overwhelming amount of information we are presented with each day. A modern application of reputation is in sorting search results, as Google does with their PageRank algorithm [77]. We see the same sort of technology being used by companies such as Facebook to provide a *personalized* experience to users, in the form of automatic selection of stories to highlight or better targeting of advertising. We are only at the tip of the iceberg for how these reputation systems can improve the quality of information we are

presented with. Imagine if these same ideas could be applied to other problems [26], such as peer review of academic papers to promote the discovery and discussion of non-mainstream works [6], or job candidate evaluation through references to discover the hidden context that each reference brings to their evaluation via their social graph and job history!

Appendices

Appendix A

Basic Difference Implementation

Perl module **WikiTrust::BasicDiff** presents the bare essence of how our differencing algorithm works.

```
package WikiTrust::BasicDiff;
use strict;
use warnings;

use constant FASTER => 1;

use WikiTrust::Tuple;
use WikiTrust::PriorityQ;
use List::Util qw(min);
use Carp;

our $VERSION = '0.01';

sub new {
  my $class = shift @_;
  my $this = bless {
     quality => \&match_quality,
     dst => [],
     minMatch => 3,
  }, $class;
  $this->init();
```

```
  return $this;
}

sub init {
  my $this = shift @_;
  $this->{heap} = WikiTrust::PriorityQ->new();
  $this->{matched_dst} = [];
}

sub set_minMatch {
    my $this = shift @_;
    $this->{minMatch} = shift @_;
}

# Parse a string into a list of words.
# For this demo, we only split on whitespace,
# but the full Ocaml version interprets wiki
# markup to better distinguish "words".
sub parse {
  my ($this, $str) = @_;
  confess "No string defined" if !defined $str;
  my @words = split(/\s+/, $str);
  return \@words;
}

# Set the destination string that we are
# trying to transform into.
sub target {
  my $this = shift @_;
  my $str = shift @_;
  $str = $this->parse($str, @_) if !ref $str;
  $this->{dst} = $str;
  return $this->{dst};
}


sub match_quality {
  my ($chunk, $k, $i1, $l1, $i2, $l2) = @_;
  my $pos1 = (2*$i1 + $k) / $l1;
  my $pos2 = (2*$i2 + $k) / $l2;
  # the closer to zero, the better
  my $q = abs($pos1 - $pos2);
  # The Heap::Priority module works much faster
  # if we use floats instead of tuples to sort
```

```
  # the entries...
  return (-$chunk*10000) + $k - $q if FASTER;
  return WikiTrust::Tuple->new(-$chunk, $k, -$q);
}

# Create a hash table indexed by word,
# which gives the list of locations where
# the word appears in the input list.
sub make_index {
  my ($this, $words) = @_;
  my $idx = {};
  for (my $i = 0; $i < @$words-1; $i++) {
    my $w1 = $words->[$i];
    my $w2 = $words->[$i+1];
    $idx->{$w1,$w2} = [] if !exists $idx->{$w1,$w2};
    push @{ $idx->{$w1,$w2} }, $i;
  }
  return $idx;
}

sub compute_heap {
  my ($this, $chunk, $w1,
    $skipmatch, $eachk, $maxk) = @_;
  my $w2 = $this->{dst};
  my $l1 = scalar(@$w1);
  my $l2 = scalar(@$w2);
  my $idx = $this->make_index($w1);
  my $prev_matches = [];
  for (my $i2 = 0; $i2 < @$w2-1; $i2++) {
    # For every unmatched word in w2,
    # find the list of matches in w1
    next if $this->{matched_dst}->[$i2];
    my $matches = $idx->{ $w2->[$i2], $w2->[$i2+1] } || [];
    foreach my $i1 (@$matches) {
      # Do we want to skip this match for some reason?
      next if $skipmatch->($chunk, $i1, $i2, $prev_matches);
      # for each match, compute all the longer strings
      # that match starting at this point.
      # Note that we already know $k == 0 is a match
      my $k = 0;
      do {
        # for each partial match, call $eachk
        $eachk->($chunk, $i1, $l1, $i2, $l2, $k+1);
        $k++;
```

```
      } while ($i1 + $k < $l1 && $i2 + $k < $l2
          && ($w1->[$i1+$k] eq $w2->[$i2+$k]));
      # And finally, call $maxk for the maximal match.
      # Note that $eachk will also have been called for
      # this same length of match.
      $maxk->($chunk, $i1, $l1, $i2, $l2, $k);
    }
    $prev_matches = $matches;
  }
}

# Given the source string we are trying to transform from,
# build the heap of matches to the destination string.
sub build_heap {
  my ($this, $chunk, $src) = @_;
  $src = $this->parse($src, @_) if !ref $src;
  $this->compute_heap($chunk, $src,
    sub { return 0; },    # never skip match
    sub {
      my ($chunk, $i1, $l1, $i2, $l2, $k) = @_;
      return if $k < $this->{minMatch};
      my $q = $this->{quality}->($chunk, $k,
        $i1, $l1, $i2, $l2);
      $this->{heap}->insert($q,
        WikiTrust::Tuple->new($chunk, $k, $i1, $i2));
    },
    sub { }
  );
}
# return a region of [start,end) which
# has $test->() false for the whole interval
sub scan_and_test {
  my ($this, $len, $test) = @_;
  return undef if $len <= 0;
  my $start = 0;
  while ($start < $len && $test->($start)) { $start++; }
  return undef if $start >= $len;
  my $end = $start+1;
  while ($end < $len && !$test->($end)) { $end++; }
  return ($start, $end);
}

sub process_best_matches {
  my ($this, $multimatch, $chunks, $chunkmatch) = @_;
```

```
  my @editScript;

  while (my $m = $this->{heap}->pop()) {
    my ($chunk, $k, $i1, $i2) = @$m;
    my $matched1 = $chunkmatch->[$chunk];
    # have any of these words already been matched?
    my ($start, $end) = $this->scan_and_test($k,
        sub { $matched1->[$i1+$_[0]]
            || $this->{matched_dst}->[$i2+$_[0]] });
    next if !defined $start;    # whole thing is matched
    if ($end - $start == $k) {
      # the whole sequence is still unmatched
      my $match =
        WikiTrust::Tuple->new('Mov', $chunk, $i1, $i2, $k);
      push @editScript, $match;
      # and mark it matched
      for (my $i = $start; $i < $end; $i++) {
        $matched1->[$i1+$i] = $match
            if !$multimatch;
        $this->{matched_dst}->[$i2+$i] = $match;
      }
    }
  }
  return \@editScript;
}

sub cover_unmatched {
  my ($this, $matched, $l, $editScript, $mode) = @_;

  my $i = 0;
  while (1) {
    my ($start, $end) = $this->scan_and_test($l,
        sub { $matched->[$i+$_[0]] });
    last if !defined $start;
    push @$editScript,
      WikiTrust::Tuple->new($mode, $i+$start, $end-$start);
    $i += $end;
    $l -= $end;
  }
}

sub replacement_scan {
    my ($this, $editScript, $matched, $len, $chunks) = @_;
```

```
}


# Compute the edit script to transform src into dst.
sub edit_diff {
  my $this = shift @_;
  my $src = shift @_;
  $src = $this->parse($src, @_) if !ref $src;

  $this->init();
  $this->build_heap(0, $src);
  my $matched_chunks = [ [] ];
  $matched_chunks->[0]->[scalar(@$src)-1] = undef;
  my $editScript = $this->process_best_matches(0, [$src],
    $matched_chunks);
  $this->replacement_scan($editScript, $this->{matched_dst},
    scalar(@{ $this->{dst} }), $matched_chunks);
  $this->cover_unmatched($matched_chunks->[0],
    scalar(@$src), $editScript, 'Del');
  $this->cover_unmatched($this->{matched_dst},
    scalar(@{ $this->{dst} }), $editScript, 'Ins');
  return $editScript;
}

1;
```

Appendix B

Faster Difference Implementation

Perl module **WikiTrust::FasterDiff** presents a modification to **WikiTrust::BasicDiff** which computes only the longest matches and their residuals.

```
package WikiTrust::FasterDiff;
# A faster diff, which assumes that longer
# matches are always prioritized before
# shorter matches.
use strict;
use warnings;

use WikiTrust::Tuple;
use WikiTrust::BasicDiff;

# Setup our baseclass; this file only has overrides
our @ISA = qw(WikiTrust::BasicDiff);

# Given the source string we are trying to transform from,
# build the heap of matches to the destination string.
sub build_heap {
  my $this = shift @_;
  my $chunk = shift @_;
  my $src = shift @_;
  $src = $this->parse($src, @_) if !ref $src;
  my %matched;
```

```
  $this->compute_heap($chunk, $src,
    sub {
      my ($chunk, $i1, $i2, $prev_matches) = @_;
      # The 'prev match' optimization:
      # return (grep { $i1 - 1 == $_ } @$prev_matches) > 0;
      return $matched{$chunk, $i1, $i2};
    },
    sub {
      # If we want to keep small matches, then we
      # can mark the match right away.  For WikiTrust,
      # we don't want small matches, so this function
      # does nothing.
      # OLD CODE:
      # my ($chunk, $i1, $l1, $i2, $l2, $k) = @_;
      ## remember that $k is the length of the match
      # $matched{$chunk,$i1+$k-1,$i2+$k-1} = 1;
    },
    sub {
      my ($chunk, $i1, $l1, $i2, $l2, $k) = @_;
      # skip short matches
      return if $k < $this->{minMatch};

      # mark the positions as matched; not necessary for
      # the 'prev matches' optimization
      foreach my $i (0..$k-1) {
        $matched{$chunk, $i1+$i,$i2+$i} = 1;
      }

      my $qfunc = $this->{quality};
      my $q = $qfunc->($chunk, $k, $i1, $l1, $i2, $l2);
      $this->{heap}->insert($q,
        WikiTrust::Tuple->new($chunk, $k, $i1, $i2));
    }
  );
}

# This is exactly the same as in the parent class, except
# for when a region has already been previously matched.
# In that case, we construct the residual matches and add
# them to the heap.  For this to work properly, we must have
# that the quality measure puts longer matches before
# shorter matches.
sub process_best_matches {
  my ($this, $multimatch, $chunks, $chunkmatch) = @_;
```

```
my $l2 = @{ $this->{dst} };

my @editScript;

while (my $m = $this->{heap}->pop()) {
  my ($chunk, $k, $i1, $i2) = @$m;
  my $w1 = $chunks->[$chunk];
  my $matched1 = $chunkmatch->[$chunk];
  my $l1 = @$w1;
  # have any of these words already been matched?
  my ($start, $end) = $this->scan_and_test($k,
      sub { $matched1->[$i1+$_[0]]
          || $this->{matched_dst}->[$i2+$_[0]] });
  next if !defined $start;    # whole thing is matched
  if ($end - $start == $k) {
    # the whole sequence is still unmatched
    my $match = WikiTrust::Tuple->new(
      'Mov', $chunk, $i1, $i2, $k
    );
    push @editScript, $match;
    # and mark it matched
    for (my $i = $start; $i < $end; $i++) {
      $matched1->[$i1+$i] = $match
          if !$multimatch;
      $this->{matched_dst}->[$i2+$i] = $match;
    }
  } else {
    # found an unmatched subregion, but it's
    # less than the size we were hoping for.
    # So we must add the smaller matches back
    # into the heap...  starting with the match
    # we just found.
    do {
      my $newK = $end - $start;
      # skip too-short matches
      if ($newK >= $this->{minMatch}) {
        my $qfunc = $this->{quality};
        my $q = $qfunc->($chunk, $newK,
          $i1+$start, $l1, $i2+$start, $l2);
        $this->{heap}->insert($q, WikiTrust::Tuple->new(
          $chunk, $newK, $i1+$start, $i2+$start
        ));
      }
```

```
            $i1 += $end;
            $i2 += $end;
            $k -= $end;
            ($start, $end) = $this->scan_and_test($k,
                sub { $matched1->[$i1+$_[0]]
                ||  $this->{matched_dst}->[$i2+$_[0]] });
        } while (defined $start);
      }
    }
    return \@editScript;
}

1;
```

Appendix C

Basic Text Tracking Implementation

Perl module **WikiTrust::BasicTextTracking** presents the bare essence of text tracking.

```
package WikiTrust::BasicTextTracking;
use strict;
use warnings;

use constant DEBUG => 0;

use WikiTrust::FasterDiff;
use WikiTrust::Word;
use Carp;

our @ISA = qw(WikiTrust::FasterDiff);

sub new {
  my $class = shift @_;
  my $self = WikiTrust::FasterDiff->new(@_);
  $self->{minMatch} = 3;
  bless $self, $class;
}

# When we parse a string into words, we actually want
# to tag each word with a revid.  Later, we will assign
# proper revids to each word.
sub parse {
```

```
  my ($this, $str, $revid) = @_;
  my $words = $this->SUPER::parse($str);
  my @words = map { WikiTrust::Word->new($_, $revid) }
    @$words;
  return \@words;
}

sub fix_author {
  my ($this, $script, $prevrevs) = @_;
  foreach my $match (@$script) {
    my $mode = shift @$match;
    confess "Bad mode: $mode" if $mode ne 'Mov';
    my ($chunk, $i1, $i2, $len) = @$match;
    # reject small matches
    ## code: next if $len < $this->{minMatch};
    for (my $i = 0; $i < $len; $i++) {
      $this->{dst}->[$i2+$i]->[1] =
        $prevrevs->[$chunk]->[$i1+$i]->[1];
    }
  }
}

sub track_text {
  my ($this, $prevrevs) = @_;
  $this->init();
  my $chunk_matches = [];

  # Build a heap of matching chunks for all the previous revs.
  for (my $chunk = 0; $chunk < @$prevrevs; $chunk++) {
    $chunk_matches->[$chunk] = [];
    my $src = $prevrevs->[$chunk];
    $this->build_heap($chunk, $src);
  }

  # And then find the best matches
  my $editScript = $this->process_best_matches(1,
    $prevrevs, $chunk_matches);
  $this->fix_author($editScript, $prevrevs);

  return $this->{dst};
}

1;
```

Appendix D

Faster Text Tracking Implementation

Perl module **WikiTrust::FasterTextTracking** presents a modification that assumes the quality function always prefers matches in more recent chunks.

```
package WikiTrust::FasterTextTracking;
# Assume that more recent chunks are always
# preferred by the quality function.
use strict;
use warnings;

use WikiTrust::BasicTextTracking;

our @ISA = qw(WikiTrust::BasicTextTracking);

# Compute the edit script to transform src into dst.  But we
# only care about mov operations, so don't compute the INS
# and DEL operations.
sub edit_diff {
  my ($this, $chunk, $src) = @_;
  # Don't call $this->init() because we want to maintain the
  # matched_dst data, which is tracking which words have
  # already been matched in the target string.  The heap
  # itself will already be empty, because
  # $this->process_best_matches() always deals with the
  # entire heap.
```

```
  $this->build_heap($chunk, $src);
  my $editScript = $this->process_best_matches(
    1, $src, []
  );
  return $editScript;
}

sub track_text {
  my ($this, $prevrevs) = @_;

  $this->init();
  # Since we prefer chunks with more liveness, we do
  # matches in a serial fashion
  for (my $chunk = 0; $chunk < @$prevrevs; $chunk++) {
    my $src = $prevrevs->[$chunk];
    my $script = $this->edit_diff($chunk, $src);
    $this->fix_author($script, $prevrevs);
  }
  return $this->{dst};
}


1;
```

Appendix E

OCaml Diff Benchmarking Code

OCaml language version of the text differencing algorithms used in the evaluation section of Chapters 3 and 4.

```
TYPE_CONV_PATH "UCSC_WIKI_RESEARCH"

open Editlist;;

type word = string
type heap_el = int * int * int
type index_t = ((word * word), int) Hashtbl_bounded.t

exception Heap_Too_Large

(** This is the maximum number of matches for a
 *  word pair that we track. If a word pair has
 *  more than this number of matches, we disregard
 *  them all, as we classify the word pair as not
 *  sufficiently distinctive.
 *)
let max_matches = 50
let max_heaplen = ref 1000
let thumper_min_copy_len = 3
```

```
module Heap = Coda.PriorityQueue
type match_quality_t = Coda.match_quality_t

(* Quality functions for matches.
 * l is the length of the match.
 * len1 and len2 are the two lengths (in number of
 *      words) of the pieces being compared.
 * i1 and i2 are the two starting points.
 * ch1_idx is the chunk number.
 * The lower the quality, the more the match is
 * considered, as elements are * removed starting
 * from the lowest from the priority queue.
 *)

let quality_live (l: int) (i1: int) (len1: int)
                 (i2: int) (len2: int) (ch1_idx: int)
    : match_quality_t =
  let i1' = (float_of_int (2 * i1 + 1)) /. 2. in
  let len1' = float_of_int len1 in
  let i2' = (float_of_int (2 * i2 + 1)) /. 2. in
  let len2' = float_of_int len2 in
  let q = abs_float ((i1' /. len1') -. (i2' /. len2')) 
  in
  (-l, -ch1_idx, q)

let quality_1 (l: int) (i1: int) (len1: int)
              (i2: int) (len2: int) (ch1_idx: int)
    : match_quality_t = (-l, ch1_idx, 0.0)

let quality_2 (l: int) (i1: int) (len1: int)
              (i2: int) (len2: int) (ch1_idx: int)
    : match_quality_t = (-l, -ch1_idx, 0.0)

let quality_3 (l: int) (i1: int) (len1: int)
              (i2: int) (len2: int) (ch1_idx: int)
    : match_quality_t = (ch1_idx, -l, 0.0)

let quality_4 (l: int) (i1: int) (len1: int)
              (i2: int) (len2: int) (ch1_idx: int)
    : match_quality_t = (-ch1_idx, -l, 0.0)

let quality_5 (l: int) (i1: int) (len1: int)
              (i2: int) (len2: int) (ch1_idx: int)
    : match_quality_t =
```

```
  let i1' = (float_of_int (2 * i1 + 1)) /. 2. in
  let len1' = float_of_int len1 in
  let i2' = (float_of_int (2 * i2 + 1)) /. 2. in
  let len2' = float_of_int len2 in
  let l' = float_of_int l in
  let correction = 0.3 *. abs_float ((i1' /. len1')
        -. (i2' /. len2')) in
  let q = l' /. (min len1' len2') -. correction
  in
  (0, -ch1_idx, 0.0 -. q)

let quality_6 (l: int) (i1: int) (len1: int)
            (i2: int) (len2: int) (ch1_idx: int)
    : match_quality_t =
  let i1' = (float_of_int (2 * i1 + 1)) /. 2. in
  let len1' = float_of_int len1 in
  let i2' = (float_of_int (2 * i2 + 1)) /. 2. in
  let len2' = float_of_int len2 in
  let l' = float_of_int l in
  let correction = 0.3 *. abs_float ((i1' /. len1')
        -. (i2' /. len2')) in
  let q = l' /. (min len1' len2') -. correction
  in
  (-l, -ch1_idx, 0.0 -. q)

let quality_7 (l: int) (i1: int) (len1: int)
            (i2: int) (len2: int) (ch1_idx: int)
    : match_quality_t =
  let i1' = (float_of_int (2 * i1 + 1)) /. 2. in
  let len1' = float_of_int len1 in
  let i2' = (float_of_int (2 * i2 + 1)) /. 2. in
  let len2' = float_of_int len2 in
  let l' = float_of_int l in
  let correction = 0.3 *. abs_float ((i1' /. len1')
        -. (i2' /. len2')) in
  let q = l' /. (min len1' len2') -. correction
  in
  (-l, ch1_idx, 0.0 -. q)

let quality_8 (l: int) (i1: int) (len1: int)
        (i2: int) (len2: int) (ch1_idx: int)
    : match_quality_t
  = quality_live l i1 len1 i2 len2 ch1_idx
```

```
let quality_9 (l: int) (i1: int) (len1: int)
        (i2: int) (len2: int) (ch1_idx: int)
    : match_quality_t =
  let i1' = (float_of_int (2 * i1 + 1)) /. 2. in
  let len1' = float_of_int len1 in
  let i2' = (float_of_int (2 * i2 + 1)) /. 2. in
  let len2' = float_of_int len2 in
  let q = abs_float ((i1' /. len1') -. (i2' /. len2'))
  in
  (-l, ch1_idx, q)


let m_quality_func = ref quality_live

let set_match_quality (i: int) =
  if i = 0 then m_quality_func := quality_live
  else if i = 1 then m_quality_func := quality_1
  else if i = 2 then m_quality_func := quality_2
  else if i = 3 then m_quality_func := quality_3
  else if i = 4 then m_quality_func := quality_4
  else if i = 5 then m_quality_func := quality_5
  else if i = 6 then m_quality_func := quality_6
  else if i = 7 then m_quality_func := quality_7
  else if i = 8 then m_quality_func := quality_live
  else if i = 9 then m_quality_func := quality_9

let make_index_diff (words: word array) : index_t =
  let len = Array.length words in
  let idx = Hashtbl_bounded.create (1 + len)
        (10 * max_matches) in
  for i = 0 to len - 2 do
    let word_tuple = (words.(i), words.(i + 1)) in
    Hashtbl_bounded.add idx word_tuple i
  done;
  idx;;

let get_matches matched idx word_tuple =
  if Hashtbl_bounded.mem idx word_tuple then begin
    let all_matches =
        Hashtbl_bounded.find_all idx word_tuple in
    let filt i = matched.(i) = 0 in
    let matches = List.filter filt all_matches in
    if (List.length all_matches) > max_matches then begin
      (* too many, so empty list *)
```

```
      Hashtbl_bounded.remove_all idx word_tuple;
      [];
    end else matches;
  end else [];;

(* This function is directly based on the
 * code included in Reichenberger1991, but
 * modified to respect the values in matched1/2.
 *)
let build_reichenberger
            (w1: word array) (w2: word array)
            matched1
            matched2
            l1 l2 =
  let editscript = ref [] in
  let idx2 = make_index_diff w2 in
  let oldPos = ref 0 in
  let addStart = ref 0 in
  let emitAdd () =
    if !addStart < !oldPos then begin
      let k = !oldPos - !addStart in
      editscript := Del (!addStart, k) :: !editscript;
      for i = !addStart to !oldPos-1 do
        matched1.(i) <- !oldPos - i;
      done;
    end
  in
  while !oldPos < l1 - 2 do
    (* for every unmatched word in w1,
     * find list of matches in w2 *)
    if matched1.(!oldPos) = 0 then begin
      let word_tuple = (w1.(!oldPos), w1.(!oldPos+1)) in
      let matches = get_matches matched2 idx2 word_tuple in
      let i1 = !oldPos in
      let heap = Heap.create () in
      let process_match (i2: int) =
        let k = ref 1 in
        while i1 + !k < l1 && i2 + !k < l2
        && w1.(i1 + !k) = w2.(i2 + !k) do
          k := !k + 1;
        done;
        let q = !m_quality_func !k i1 l1 i2 l2 0 in
        ignore (Heap.add heap (!k, i1, i2) q);
      in
```

```
      List.iter process_match matches;
      if not (Heap.is_empty heap) then begin
        let m = Heap.take heap in
        let (copyLen, i1', copyStart) = m.Heap.contents in
        if copyLen >= thumper_min_copy_len then begin
          emitAdd ();
          editscript := Mov (!oldPos, copyStart, copyLen)
            :: !editscript;
          for i = 0 to copyLen - 1 do
            let nextMatch = copyLen - i in
            matched2.(copyStart + i) <- nextMatch;
            matched1.(!oldPos + i) <- nextMatch;
          done;
          oldPos := !oldPos + copyLen;
          addStart := !oldPos;
        end else
          oldPos := !oldPos + 1;
      end else
        oldPos := !oldPos + 1;
    end else begin
      if !addStart < !oldPos then emitAdd ();
      oldPos := !oldPos + matched1.(!oldPos);
      addStart := !oldPos;
    end;
  done;
  (* we can skip the final emitAdd, since
   * cover_unmatched will cleanup *)
  editscript;;

let compute_heap
            (w1: word array) (w2: word array)
            matched1 matched2
            skipmatch eachk maxk =
  let l1 = Array.length w1 in
  let l2 = Array.length w2 in
  let idx1 = make_index_diff w1 in
  let prev_matches = ref [] in
  let i2 = ref 0 in
  while !i2 < l2 - 1 do
    let skip = matched2.(!i2) in
    if skip = 0 then begin
      let word_tuple = (w2.(!i2), w2.(!i2+1)) in
      let matches = get_matches matched1 idx1 word_tuple in
      let process_match (i1: int) =
```

```
        if not (skipmatch i1 !i2 prev_matches) then begin
          let k = ref 1 in
          eachk i1 l1 !i2 l2 !k;
          while i1 + !k < l1 && !i2 + !k < l2
          && w1.(i1 + !k) = w2.(!i2 + !k) do
            eachk i1 l1 !i2 l2 (!k + 1);
            k := !k + 1;
          done;
          maxk i1 l1 !i2 l2 !k
        end
      in
      List.iter process_match matches;
      prev_matches := matches;
    end else prev_matches := [];
    i2 := !i2 + (max 1 skip)
  done;;

(**
 * This version only puts the longest matches in the heap,
 * and it only checks the list of previous matches
 * from the last match to see if a new match
 * should be added.
 *)
let build_heap_fastpm
            (w1: word array) (w2: word array)
            matched1 matched2 =
  let heap = Heap.create () in
  let skipmatch i1 i2 prev_matches =
    (* if (i1-1) is in prev_matches, then we've
     * already investigated a longer match
     * starting at (i1-1, i2-1) (or even earlier),
     * so we can skip this one *)
    List.mem (i1 - 1) !prev_matches
  in
  let eachk i1 l1 i2 l2 k = () in
  let maxk i1 l1 i2 l2 k =
    if k >= thumper_min_copy_len then begin
      let q = !m_quality_func k i1 l1 i2 l2 0 in
      ignore (Heap.add heap (k, i1, i2) q);
    end
  in
  compute_heap w1 w2 matched1 matched2 skipmatch eachk maxk;
  heap
```

```
(** This version only puts the longest match into
 * the heap, but uses a hashtable to keep track of
 * what matches have been made, rather than just
 * checking the previous list of matches.
 *)
let build_heap_fasthash
            (w1: word array) (w2: word array)
            matched1 matched2 =
  let len1 = Array.length w1 in
  let len2 = Array.length w2 in
  let matched = Hashtbl.create (len1 + len2) in
  let heap = Heap.create () in
  let skipmatch i1 i2 prev_matches =
    let idx = (i1, i2) in
    try Hashtbl.find matched idx
    with Not_found -> false
  in
  let eachk i1 l1 i2 l2 k = () in
  let maxk i1 l1 i2 l2 k =
    if k >= thumper_min_copy_len then begin
      for i = 0 to (k-1) do
        let idx = (i1+i, i2+i) in
        Hashtbl.replace matched idx true
      done;
      let q = !m_quality_func k i1 l1 i2 l2 0 in
      ignore (Heap.add heap (k, i1, i2) q);
    end
  in
  compute_heap w1 w2 matched1 matched2 skipmatch eachk maxk;
  heap

(** This version of heap building is the slowest,
 * because it includes every single possible match
 * in the heap, not just the longest possible
 * match.  This ends up using a very large amount
 * of memory; on the order of gigabytes, versus
 * the roughly 500MB that the longest-match
 * version uses for the PAN2010 evaluation.
 *)
let build_heap_slow
            (w1: word array) (w2: word array)
            matched1 matched2 =
  let heap = Heap.create () in
  let skipmatch i1 i2 prev_matches = false
```

```
  in
  let eachk i1 l1 i2 l2 k =
    if k >= thumper_min_copy_len then begin
      let q = !m_quality_func k i1 l1 i2 l2 0 in
      ignore (Heap.add heap (k, i1, i2) q);
    end
  in
  let maxk i1 l1 i2 l2 k =
    (* already done in eachk *)
    ()
  in
  compute_heap w1 w2 matched1 matched2 skipmatch eachk maxk;
  heap

(**
 * Find a region where 'test' is 0,
 * and return the bounds of that region.
 * The 'test' parameter otherwise tells
 * us an upper bound on how far forward
 * we can safely skip.
 *)
let scan_and_test len test =
  let rec find_start curstart =
    if curstart >= len then curstart
    else begin
      let incr = test curstart in
      if incr = 0 then curstart
      else find_start (curstart + incr)
    end
  in
  let rec find_finish curend =
    if curend >= len then curend
    else begin
      let incr = test curend in
      if incr > 0 then curend
      else find_finish (curend + 1)
    end
  in
  let start = find_start 0 in
  let finish = find_finish (start + 1) in
  if start >= len then (-1, -1)
  else (start, finish)
  ;;
```

```
let process_best_matches heap matched1 matched2 l1 l2 =
  let editscript = ref [] in
  let record_match i1 i2 k =
    editscript := Mov (i1, i2, k) :: !editscript;
    for i = 0 to k-1 do
      let nextMatch = k - i in
      matched1.(i1 + i) <- nextMatch;
      matched2.(i2 + i) <- nextMatch;
    done
  in
  let make_test i1 i2 =
    let is_matched offset =
      max matched1.(i1 + offset) matched2.(i2 + offset)
    in
    is_matched
  in
  let rec add_smaller i1 i2 start finish limit =
    if start >= 0 then begin
      let k = finish - start in
      let i1 = i1 + start in
      let i2 = i2 + start in
      if k >= thumper_min_copy_len then begin
        let q = !m_quality_func k i1 l1 i2 l2 0 in
        ignore (Heap.add heap (k, i1, i2) q)
      end;
      (* compute range for next possible sub-match *)
      let i1 = i1 + k in
      let i2 = i2 + k in
      let limit = limit - finish in
      let is_matched = make_test i1 i2 in
      let (start, finish) = scan_and_test limit is_matched in
      add_smaller i1 i2 start finish limit
    end else ()
  in
  let heaplen = Heap.length heap in
  if heaplen > !max_heaplen + 1000 then begin
    max_heaplen := heaplen;
    print_endline
      (Printf.sprintf "new␣max␣heap:␣%d" !max_heaplen);
    flush stdout;
  end;
  if heaplen > 1000000 then begin
    raise Heap_Too_Large;
  end;
```

```
  while not (Heap.is_empty heap) do
    let m = Heap.take heap in
    let (k, i1,  i2) = m.Heap.contents in
    let (start, finish) = scan_and_test k (make_test i1 i2) in
    if start >= 0 then begin
      if finish - start = k then begin
        (* the whole sequence is still unmatched *)
        record_match i1 i2 k
      end else begin
        (* found an unmatched subregion, but it's for less
         * than the size we were hoping for.  So we must add
         * the smaller matches back into the heap... starting
         * with the match we just found. *)
        add_smaller i1 i2 start finish k
      end;
    end;
  done;
  editscript
  ;;

let cover_unmatched matched len editScript op =
  let i = ref 0 in
  let l = ref len in
  let complete = ref false in
  while not !complete do
    let test x = matched.(!i + x) in
    let (start, finish) = scan_and_test !l test in
    if start >= 0 then begin
      let tuple = op (!i + start) (finish - start) in
      editScript := tuple :: !editScript;
      i := !i + finish;
      l := !l - finish;
    end
      else complete := true
  done;
  editScript
  ;;

let match_endpoint (w1: word array) (w2: word array)
            matched1 matched2 xform1 xform2 =
  let l1 = Array.length w1 in
  let l2 = Array.length w2 in
  let k = min l1 l2 in
  let rec find_first_nonmatch x =
```

```
    if x >= k then k
    else begin
      let i1 = xform1 x in
      let i2 = xform2 x in
      if matched1.(i1) > 0 || matched2.(i2) > 0
        || w1.(i1) <> w2.(i2)
      then x
      else find_first_nonmatch (x + 1)
    end
  in
  let nonmatch = find_first_nonmatch 0 in
  if nonmatch > 0 then begin
    let endpoint1 = max (xform1 (-1)) nonmatch in
    let endpoint2 = max (xform2 (-1)) nonmatch in
    for i = 0 to nonmatch - 1 do
      matched1.(xform1 i) <- abs (endpoint1 - i);
      matched2.(xform2 i) <- abs (endpoint2 - i);
    done;
    let beginpt1 = min (xform1 0) (xform1 (nonmatch-1)) in
    let beginpt2 = min (xform2 0) (xform2 (nonmatch-1)) in
    [ Mov (beginpt1, beginpt2, nonmatch) ];
  end else [ ]
  ;;

let match_header (w1: word array) (w2: word array)
            matched1 matched2 =
  let xform1 x = x in
  let xform2 x = x in
  match_endpoint w1 w2 matched1 matched2 xform1 xform2
  ;;

let match_trailer (w1: word array) (w2: word array)
        matched1 matched2 =
  let l1 = Array.length w1 in
  let l2 = Array.length w2 in
  let xform1 x = l1 - x - 1 in
  let xform2 x = l2 - x - 1 in
  match_endpoint w1 w2 matched1 matched2 xform1 xform2
  ;;

let match_nothing (w1: word array) (w2: word array)
        matched1 matched2 = [ ]
let makeDel i l = Del (i, l)
let makeIns i l = Ins (i, l)
```

```
let core_diff w1 w2 mkHeader mkTrailer mkEditScript =
  let l1 = Array.length w1 in
  let l2 = Array.length w2 in
  let matched1 = Array.make l1 0 in
  let matched2 = Array.make l2 0 in
  let header = mkHeader w1 w2 matched1 matched2 in
  let trailer = mkTrailer w1 w2 matched1 matched2 in
  let editScript =
      mkEditScript w1 w2 matched1 matched2 l1 l2 in
  let editScript = cover_unmatched matched1 l1
    editScript makeDel in
  let editScript = cover_unmatched matched2 l2
    editScript makeIns in
  header @ !editScript @ trailer

let diff_1 (w1: word array) (w2: word array) =
  let myCore w1 w2 matched1 matched2 l1 l2 =
    build_reichenberger w1 w2 matched1 matched2 l1 l2 in
  core_diff w1 w2
        match_nothing match_nothing
        myCore

let diff_2 (w1: word array) (w2: word array) =
  let myCore w1 w2 matched1 matched2 l1 l2 =
    build_reichenberger w1 w2 matched1 matched2 l1 l2 in
  core_diff w1 w2
        match_header match_trailer
        myCore

let diff_3 (w1: word array) (w2: word array) =
  let myCore w1 w2 matched1 matched2 l1 l2 =
    let heap =
        build_heap_fasthash w1 w2 matched1 matched2 in
    process_best_matches heap matched1 matched2 l1 l2
  in
  core_diff w1 w2
        match_header match_trailer
        myCore

let diff_4 (w1: word array) (w2: word array) =
  let myCore w1 w2 matched1 matched2 l1 l2 =
    let heap = build_heap_fastpm w1 w2 matched1 matched2 in
    process_best_matches heap matched1 matched2 l1 l2
```

```
  in
  core_diff w1 w2
        match_nothing match_nothing
        myCore

let diff_5 (w1: word array) (w2: word array) =
  let myCore w1 w2 matched1 matched2 l1 l2 =
    let heap = build_heap_fastpm w1 w2 matched1 matched2 in
    process_best_matches heap matched1 matched2 l1 l2
  in
  core_diff w1 w2
        match_header match_trailer
        myCore

let diff_8 (w1: word array) (w2: word array) =
  let myCore w1 w2 matched1 matched2 l1 l2 =
    let heap = build_heap_fasthash w1 w2 matched1 matched2 in
    process_best_matches heap matched1 matched2 l1 l2
  in
  core_diff w1 w2
        match_nothing match_nothing
        myCore

let diff_9 (w1: word array) (w2: word array) =
  let myCore w1 w2 matched1 matched2 l1 l2 =
    let heap = build_heap_slow w1 w2 matched1 matched2 in
    process_best_matches heap matched1 matched2 l1 l2
  in
  core_diff w1 w2
        match_header match_trailer
        myCore



let diff_func = ref diff_1

let set_diff (i: int) =
  if i = 1 then diff_func := diff_1
  else if i = 2 then diff_func := diff_2
  else if i = 3 then diff_func := diff_3
  else if i = 4 then diff_func := diff_4
  else if i = 5 then diff_func := diff_5
  else if i = 8 then diff_func := diff_8
  else if i = 9 then diff_func := diff_9
```

```
let edit_diff (words1: word array) (words2: word array)
  : edit list = !diff_func words1 words2
```

Appendix F

Edit Longevity Parameter Rankings

Presented here is the complete rankings of each parameter variation explored for the evaluation of edit longevity presented in Chapter 4. In addition to the fields described in Chapter 4, we also collected some additional measures of the work being done by the various combinations of parameters.

Num Revs is how many PAN-WVC-10 revisions predictions were made for. We would like this to be as high as possible, but WikiTrust will skip calculating a quality for a revision if it believes the edit distance from the previous revision is 0; this is because the calculation includes a division. In hindsight, it might be better to inspect the edit script directly to determine when there is no change compare from the previous revision. When there is no change between revisions, each difference algorithm will identify a single **Mov** operation. This begs the question of how to compute the edit longevity when the edit distance is zero; a reasonable choice would be setting the edit longevity to zero when the edit distance is zero. This choice indicates that there is no bias toward believing that the revision is either high- or low-quality.

Another measure of work (that is strongly related to the number of revisions for which predictions are made) is the number of triangles which are evaluated, **Total Tri-**

angles, as illustrated by Figure 4.6. Each side of the triangle is calculated in the computation for edit longevity (see Eq. 4.4). The column **Bad Triangles** counts how many of the triangles evaluated fail to satisfy the triangle inequality.

With respect to the triangle inequality, the data shows that it is not a necessary property in order to achieve good performance. In fact, the combination of using a greedy approach to generating the edit script (which might not generate the shortest edit script) and our unusual definitions of edit distance make the triangle inequality an unlikely property. See [89] for discussion about when the triangle inequality might hold, especially in the more difficult case of including transpositions.

Diff	Match Quality	Edit Dist	PR-AUC	ROC-AUC	Num Revs	Run Time	Total Triangles	Bad Triangles
diff2	mq9	ed5	47.279%	91.881%	27,730	96m	852,040	180,624
diff2	mq8	ed5	47.279%	91.881%	27,730	96m	852,040	180,624
diff2	mq7	ed5	47.279%	91.881%	27,730	96m	852,040	180,624
diff2	mq6	ed5	47.279%	91.881%	27,730	97m	852,040	180,624
diff2	mq4	ed5	47.258%	91.872%	27,730	96m	852,040	182,553
diff2	mq3	ed5	47.258%	91.872%	27,730	96m	852,040	182,553
diff2	mq2	ed5	47.258%	91.872%	27,730	96m	852,040	182,553
diff2	mq1	ed5	47.258%	91.872%	27,730	96m	852,040	182,553
diff5	mq5	ed5	47.231%	91.955%	27,730	167m	852,040	171,265
diff3	mq5	ed5	47.231%	91.955%	27,730	217m	852,040	171,271
diff8	mq5	ed5	47.198%	91.905%	27,731	349m	852,043	165,804
diff4	mq5	ed5	47.198%	91.905%	27,731	255m	852,043	165,812
diff5	mq9	ed5	47.192%	91.941%	27,730	165m	852,040	167,277
diff5	mq8	ed5	47.192%	91.941%	27,730	165m	852,040	167,277
diff5	mq7	ed5	47.192%	91.941%	27,730	166m	852,040	167,277
diff5	mq6	ed5	47.192%	91.941%	27,730	166m	852,040	167,277
diff3	mq9	ed5	47.192%	91.941%	27,730	215m	852,040	167,277
diff3	mq8	ed5	47.192%	91.941%	27,730	215m	852,040	167,277
diff3	mq7	ed5	47.192%	91.941%	27,730	216m	852,040	167,277
diff3	mq6	ed5	47.192%	91.941%	27,730	216m	852,040	167,277
diff2	mq5	ed5	47.171%	91.855%	27,730	97m	852,040	183,244
diff3	mq4	ed5	47.152%	91.892%	27,730	203m	852,040	171,073
diff3	mq3	ed5	47.152%	91.892%	27,730	203m	852,040	171,073
diff3	mq2	ed5	47.152%	91.892%	27,730	202m	852,040	171,073
diff3	mq1	ed5	47.152%	91.892%	27,730	203m	852,040	171,073
diff8	mq9	ed5	47.140%	91.887%	27,731	346m	852,043	161,757
diff8	mq8	ed5	47.140%	91.887%	27,731	346m	852,043	161,757

Diff	Match Quality	Edit Dist	PR-AUC	ROC-AUC	Num Revs	Run Time	Total Triangles	Bad Triangles
diff8	mq7	ed5	47.140%	91.887%	27,731	348m	852,043	161,757
diff8	mq6	ed5	47.140%	91.887%	27,731	348m	852,043	161,757
diff4	mq9	ed5	47.140%	91.887%	27,731	252m	852,043	161,764
diff4	mq8	ed5	47.140%	91.887%	27,731	252m	852,043	161,764
diff4	mq7	ed5	47.140%	91.887%	27,731	254m	852,043	161,764
diff4	mq6	ed5	47.140%	91.887%	27,731	254m	852,043	161,764
diff4	mq4	ed5	47.124%	91.830%	27,731	227m	852,043	165,061
diff4	mq3	ed5	47.124%	91.830%	27,731	227m	852,043	165,061
diff4	mq2	ed5	47.124%	91.830%	27,731	227m	852,043	165,061
diff4	mq1	ed5	47.124%	91.830%	27,731	227m	852,043	165,061
diff8	mq4	ed5	47.111%	91.829%	27,731	320m	852,043	164,949
diff8	mq3	ed5	47.111%	91.829%	27,731	320m	852,043	164,949
diff8	mq2	ed5	47.111%	91.829%	27,731	320m	852,043	164,949
diff8	mq1	ed5	47.111%	91.829%	27,731	320m	852,043	164,949
diff5	mq4	ed5	47.095%	91.882%	27,730	153m	852,040	171,159
diff5	mq3	ed5	47.095%	91.882%	27,730	153m	852,040	171,159
diff5	mq2	ed5	47.095%	91.882%	27,730	153m	852,040	171,159
diff5	mq1	ed5	47.095%	91.882%	27,730	153m	852,040	171,159
diff6	mq4	ed5	47.041%	91.804%	27,730	273m	852,040	165,048
diff6	mq3	ed5	47.041%	91.804%	27,730	273m	852,040	165,048
diff6	mq2	ed5	47.041%	91.804%	27,730	272m	852,040	165,048
diff6	mq1	ed5	47.041%	91.804%	27,730	273m	852,040	165,048
diff1	mq9	ed5	47.037%	91.796%	27,731	97m	852,043	160,927
diff1	mq8	ed5	47.037%	91.796%	27,731	97m	852,043	160,927
diff1	mq7	ed5	47.037%	91.796%	27,731	97m	852,043	160,927
diff1	mq6	ed5	47.037%	91.796%	27,731	97m	852,043	160,927
diff1	mq5	ed5	47.035%	91.781%	27,731	97m	852,043	165,998

Diff	Match Quality	Edit Dist	PR-AUC	ROC-AUC	Num Revs	Run Time	Total Triangles	Bad Triangles
diff1	mq4	ed5	46.990%	91.776%	27,731	96m	852,043	159,556
diff1	mq3	ed5	46.990%	91.776%	27,731	96m	852,043	159,556
diff1	mq2	ed5	46.990%	91.776%	27,731	96m	852,043	159,556
diff1	mq1	ed5	46.990%	91.776%	27,731	96m	852,043	159,556
diff6	mq9	ed5	46.943%	91.842%	27,730	476m	852,040	157,362
diff6	mq8	ed5	46.943%	91.842%	27,730	477m	852,040	157,362
diff6	mq7	ed5	46.943%	91.842%	27,730	489m	852,040	157,362
diff6	mq6	ed5	46.943%	91.842%	27,730	488m	852,040	157,362
diff7	mq4	ed5	46.583%	91.742%	27,730	156m	852,040	171,397
diff7	mq3	ed5	46.583%	91.742%	27,730	156m	852,040	171,397
diff7	mq2	ed5	46.583%	91.742%	27,730	156m	852,040	171,397
diff7	mq1	ed5	46.583%	91.742%	27,730	156m	852,040	171,397
diff6	mq5	ed5	46.552%	91.709%	27,730	493m	852,040	164,461
diff7	mq9	ed5	46.539%	91.761%	27,730	253m	852,040	162,922
diff7	mq8	ed5	46.539%	91.761%	27,730	253m	852,040	162,922
diff7	mq7	ed5	46.539%	91.761%	27,730	259m	852,040	162,922
diff7	mq6	ed5	46.539%	91.761%	27,730	259m	852,040	162,922
diff7	mq5	ed5	46.100%	91.648%	27,730	261m	852,040	170,912
diff1	mq9	ed3	44.464%	91.706%	27,662	96m	850,603	39,032
diff1	mq8	ed3	44.464%	91.706%	27,662	96m	850,603	39,032
diff1	mq7	ed3	44.464%	91.706%	27,662	96m	850,603	39,032
diff1	mq6	ed3	44.464%	91.706%	27,662	96m	850,603	39,032
diff1	mq5	ed3	44.456%	91.718%	27,663	96m	850,634	42,117
diff1	mq4	ed3	44.450%	91.704%	27,664	95m	850,621	39,556
diff1	mq3	ed3	44.450%	91.704%	27,664	95m	850,621	39,556
diff1	mq2	ed3	44.450%	91.704%	27,664	95m	850,621	39,556
diff1	mq1	ed3	44.450%	91.704%	27,664	95m	850,621	39,556

Diff	Match Quality	Edit Dist	PR-AUC	ROC-AUC	Num Revs	Run Time	Total Triangles	Bad Triangles
diff8	mq5	ed3	44.240%	91.717%	27,685	348m	851,105	29,473
diff4	mq5	ed3	44.240%	91.717%	27,685	254m	851,105	29,471
diff2	mq5	ed3	44.155%	91.657%	27,662	96m	850,670	54,806
diff8	mq9	ed3	44.141%	91.683%	27,685	345m	851,087	25,967
diff8	mq8	ed3	44.141%	91.683%	27,685	345m	851,087	25,967
diff8	mq7	ed3	44.141%	91.683%	27,685	347m	851,087	25,967
diff8	mq6	ed3	44.141%	91.683%	27,685	347m	851,087	25,967
diff4	mq9	ed3	44.141%	91.683%	27,685	251m	851,087	25,967
diff4	mq8	ed3	44.141%	91.683%	27,685	251m	851,087	25,967
diff4	mq7	ed3	44.141%	91.683%	27,685	253m	851,087	25,967
diff4	mq6	ed3	44.141%	91.683%	27,685	253m	851,087	25,967
diff4	mq4	ed3	44.122%	91.677%	27,684	227m	851,079	26,633
diff4	mq3	ed3	44.122%	91.677%	27,684	226m	851,079	26,633
diff4	mq2	ed3	44.122%	91.677%	27,684	226m	851,079	26,633
diff4	mq1	ed3	44.122%	91.677%	27,684	226m	851,079	26,633
diff6	mq4	ed3	44.106%	91.735%	27,669	272m	850,636	20,315
diff6	mq3	ed3	44.106%	91.735%	27,669	272m	850,636	20,315
diff6	mq2	ed3	44.106%	91.735%	27,669	271m	850,636	20,315
diff6	mq1	ed3	44.106%	91.735%	27,669	271m	850,636	20,315
diff8	mq4	ed3	44.099%	91.674%	27,684	319m	851,079	26,594
diff8	mq3	ed3	44.099%	91.674%	27,684	319m	851,079	26,594
diff8	mq2	ed3	44.099%	91.674%	27,684	319m	851,079	26,594
diff8	mq1	ed3	44.099%	91.674%	27,684	319m	851,079	26,594
diff5	mq5	ed3	44.067%	91.742%	27,688	166m	851,218	38,179
diff3	mq5	ed3	44.067%	91.742%	27,688	216m	851,218	38,180
diff7	mq9	ed3	44.052%	91.758%	27,635	252m	850,060	46,740
diff7	mq8	ed3	44.052%	91.758%	27,635	252m	850,060	46,740

Diff	Match Quality	Edit Dist	PR-AUC	ROC-AUC	Num Revs	Run Time	Total Triangles	Bad Triangles
diff7	mq7	ed3	44.052%	91.758%	27,635	258m	850,060	46,740
diff7	mq6	ed3	44.052%	91.758%	27,635	258m	850,060	46,740
diff6	mq9	ed3	44.042%	91.737%	27,669	476m	850,636	18,915
diff6	mq8	ed3	44.042%	91.737%	27,669	475m	850,636	18,915
diff6	mq7	ed3	44.042%	91.737%	27,669	488m	850,636	18,915
diff6	mq6	ed3	44.042%	91.737%	27,669	488m	850,636	18,915
diff2	mq4	ed3	44.022%	91.635%	27,662	95m	850,653	54,059
diff2	mq3	ed3	44.022%	91.635%	27,662	95m	850,653	54,059
diff2	mq2	ed3	44.022%	91.635%	27,662	95m	850,653	54,059
diff2	mq1	ed3	44.022%	91.635%	27,662	95m	850,653	54,059
diff2	mq9	ed3	44.020%	91.637%	27,662	95m	850,670	52,077
diff2	mq8	ed3	44.020%	91.637%	27,662	95m	850,670	52,077
diff2	mq7	ed3	44.020%	91.637%	27,662	96m	850,670	52,077
diff2	mq6	ed3	44.020%	91.637%	27,662	96m	850,670	52,077
diff7	mq4	ed3	43.988%	91.769%	27,635	156m	850,060	48,381
diff7	mq3	ed3	43.988%	91.769%	27,635	156m	850,060	48,381
diff7	mq2	ed3	43.988%	91.769%	27,635	155m	850,060	48,381
diff7	mq1	ed3	43.988%	91.769%	27,635	155m	850,060	48,381
diff5	mq9	ed3	43.970%	91.708%	27,688	164m	851,201	34,742
diff5	mq8	ed3	43.970%	91.708%	27,688	164m	851,201	34,742
diff5	mq7	ed3	43.970%	91.708%	27,688	165m	851,201	34,742
diff5	mq6	ed3	43.970%	91.708%	27,688	165m	851,201	34,742
diff5	mq4	ed3	43.970%	91.701%	27,688	152m	851,201	35,571
diff5	mq3	ed3	43.970%	91.701%	27,688	152m	851,201	35,571
diff5	mq2	ed3	43.970%	91.701%	27,688	152m	851,201	35,571
diff5	mq1	ed3	43.970%	91.701%	27,688	152m	851,201	35,571
diff3	mq9	ed3	43.970%	91.708%	27,688	214m	851,201	34,742

Diff	Match Quality	Edit Dist	PR-AUC	ROC-AUC	Num Revs	Run Time	Total Triangles	Bad Triangles
diff3	mq8	ed3	43.970%	91.708%	27,688	214m	851,201	34,742
diff3	mq7	ed3	43.970%	91.708%	27,688	215m	851,201	34,742
diff3	mq6	ed3	43.970%	91.708%	27,688	215m	851,201	34,742
diff3	mq4	ed3	43.935%	91.697%	27,688	202m	851,201	35,520
diff3	mq3	ed3	43.935%	91.697%	27,688	202m	851,201	35,520
diff3	mq2	ed3	43.935%	91.697%	27,688	202m	851,201	35,520
diff3	mq1	ed3	43.935%	91.697%	27,688	201m	851,201	35,520
diff7	mq5	ed3	43.889%	91.726%	27,635	260m	850,060	53,900
diff6	mq5	ed3	43.774%	91.654%	27,670	492m	850,654	24,483
diff7	mq9	ed1	43.033%	91.524%	27,635	252m	850,060	52,221
diff7	mq8	ed1	43.033%	91.524%	27,635	252m	850,060	52,221
diff7	mq7	ed1	43.033%	91.524%	27,635	258m	850,060	52,221
diff7	mq6	ed1	43.033%	91.524%	27,635	258m	850,060	52,221
diff7	mq4	ed1	43.015%	91.539%	27,635	156m	850,060	53,909
diff7	mq3	ed1	43.015%	91.539%	27,635	156m	850,060	53,909
diff7	mq2	ed1	43.015%	91.539%	27,635	155m	850,060	53,909
diff7	mq1	ed1	43.015%	91.539%	27,635	155m	850,060	53,909
diff7	mq5	ed1	43.002%	91.503%	27,635	260m	850,060	59,873
diff6	mq4	ed1	42.891%	91.456%	27,669	272m	850,636	22,760
diff6	mq3	ed1	42.891%	91.456%	27,669	272m	850,636	22,760
diff6	mq2	ed1	42.891%	91.456%	27,669	271m	850,636	22,760
diff6	mq1	ed1	42.891%	91.456%	27,669	271m	850,636	22,760
diff6	mq9	ed1	42.843%	91.460%	27,669	475m	850,636	21,287
diff6	mq8	ed1	42.843%	91.460%	27,669	475m	850,636	21,287
diff6	mq7	ed1	42.843%	91.460%	27,669	488m	850,636	21,287
diff6	mq6	ed1	42.843%	91.460%	27,669	487m	850,636	21,287
diff6	mq5	ed1	42.744%	91.399%	27,670	492m	850,654	27,392

Diff	Match Quality	Edit Dist	PR-AUC	ROC-AUC	Num Revs	Run Time	Total Triangles	Bad Triangles
diff2	mq5	ed1	42.665%	91.336%	27,662	96m	850,670	63,951
diff1	mq5	ed1	42.662%	91.346%	27,663	96m	850,634	48,956
diff5	mq5	ed1	42.617%	91.400%	27,688	166m	851,218	43,592
diff3	mq5	ed1	42.617%	91.400%	27,688	216m	851,218	43,593
diff5	mq4	ed1	42.561%	91.366%	27,688	152m	851,201	40,226
diff5	mq3	ed1	42.561%	91.366%	27,688	152m	851,201	40,226
diff5	mq2	ed1	42.561%	91.366%	27,688	152m	851,201	40,226
diff5	mq1	ed1	42.561%	91.366%	27,688	152m	851,201	40,226
diff3	mq4	ed1	42.541%	91.362%	27,688	202m	851,201	40,153
diff3	mq3	ed1	42.541%	91.362%	27,688	202m	851,201	40,153
diff3	mq2	ed1	42.541%	91.362%	27,688	202m	851,201	40,153
diff3	mq1	ed1	42.541%	91.362%	27,688	201m	851,201	40,153
diff5	mq9	ed1	42.535%	91.368%	27,688	164m	851,201	39,356
diff5	mq8	ed1	42.535%	91.368%	27,688	164m	851,201	39,356
diff5	mq7	ed1	42.535%	91.368%	27,688	165m	851,201	39,356
diff5	mq6	ed1	42.535%	91.368%	27,688	165m	851,201	39,356
diff3	mq9	ed1	42.535%	91.368%	27,688	214m	851,201	39,356
diff3	mq8	ed1	42.535%	91.368%	27,688	214m	851,201	39,356
diff3	mq7	ed1	42.535%	91.368%	27,688	215m	851,201	39,356
diff3	mq6	ed1	42.535%	91.368%	27,688	215m	851,201	39,356
diff1	mq4	ed1	42.512%	91.290%	27,664	95m	850,621	45,245
diff1	mq3	ed1	42.512%	91.290%	27,664	95m	850,621	45,245
diff1	mq2	ed1	42.512%	91.290%	27,664	95m	850,621	45,245
diff1	mq1	ed1	42.512%	91.290%	27,664	95m	850,621	45,245
diff1	mq9	ed1	42.502%	91.290%	27,662	96m	850,603	44,723
diff1	mq8	ed1	42.502%	91.290%	27,662	96m	850,603	44,723
diff1	mq7	ed1	42.502%	91.290%	27,662	96m	850,603	44,723

Diff	Match Quality	Edit Dist	PR-AUC	ROC-AUC	Num Revs	Run Time	Total Triangles	Bad Triangles
diff1	mq6	ed1	42.502%	91.290%	27,662	96m	850,603	44,723
diff2	mq9	ed1	42.460%	91.294%	27,662	95m	850,670	60,129
diff2	mq8	ed1	42.460%	91.294%	27,662	95m	850,670	60,129
diff2	mq7	ed1	42.460%	91.294%	27,662	96m	850,670	60,129
diff2	mq6	ed1	42.460%	91.294%	27,662	96m	850,670	60,129
diff2	mq4	ed1	42.437%	91.285%	27,662	95m	850,653	62,185
diff2	mq3	ed1	42.437%	91.285%	27,662	95m	850,653	62,185
diff2	mq2	ed1	42.437%	91.285%	27,662	95m	850,653	62,185
diff2	mq1	ed1	42.437%	91.285%	27,662	95m	850,653	62,185
diff8	mq5	ed1	42.415%	91.302%	27,685	348m	851,105	34,144
diff4	mq5	ed1	42.415%	91.302%	27,685	254m	851,105	34,142
diff4	mq4	ed1	42.355%	91.268%	27,684	226m	851,079	30,487
diff4	mq3	ed1	42.355%	91.268%	27,684	226m	851,079	30,487
diff4	mq2	ed1	42.355%	91.268%	27,684	226m	851,079	30,487
diff4	mq1	ed1	42.355%	91.268%	27,684	226m	851,079	30,487
diff8	mq9	ed1	42.344%	91.271%	27,685	347m	851,087	29,801
diff8	mq8	ed1	42.344%	91.271%	27,685	345m	851,087	29,801
diff8	mq7	ed1	42.344%	91.271%	27,685	347m	851,087	29,801
diff8	mq6	ed1	42.344%	91.271%	27,685	347m	851,087	29,801
diff4	mq9	ed1	42.344%	91.271%	27,685	251m	851,087	29,801
diff4	mq8	ed1	42.344%	91.271%	27,685	251m	851,087	29,801
diff4	mq7	ed1	42.344%	91.271%	27,685	253m	851,087	29,801
diff4	mq6	ed1	42.344%	91.271%	27,685	253m	851,087	29,801
diff8	mq4	ed1	42.332%	91.264%	27,684	319m	851,079	30,450
diff8	mq3	ed1	42.332%	91.264%	27,684	319m	851,079	30,450
diff8	mq2	ed1	42.332%	91.264%	27,684	319m	851,079	30,450
diff8	mq1	ed1	42.332%	91.264%	27,684	319m	851,079	30,450

Diff	Match Quality	Edit Dist	PR-AUC	ROC-AUC	Num Revs	Run Time	Total Triangles	Bad Triangles
diff8	mq5	ed4	39.279%	88.449%	27,731	348m	852,043	26,289
diff4	mq5	ed4	39.279%	88.449%	27,731	255m	852,043	26,287
diff8	mq9	ed4	39.251%	88.431%	27,731	354m	852,043	22,655
diff8	mq8	ed4	39.251%	88.431%	27,731	345m	852,043	22,655
diff8	mq7	ed4	39.251%	88.431%	27,731	347m	852,043	22,655
diff8	mq6	ed4	39.251%	88.431%	27,731	347m	852,043	22,655
diff4	mq9	ed4	39.251%	88.431%	27,731	251m	852,043	22,655
diff4	mq8	ed4	39.251%	88.431%	27,731	251m	852,043	22,655
diff4	mq7	ed4	39.251%	88.431%	27,731	253m	852,043	22,655
diff4	mq6	ed4	39.251%	88.431%	27,731	253m	852,043	22,655
diff4	mq4	ed4	39.187%	88.401%	27,731	227m	852,043	25,749
diff4	mq3	ed4	39.187%	88.401%	27,731	226m	852,043	25,749
diff4	mq2	ed4	39.187%	88.401%	27,731	227m	852,043	25,749
diff4	mq1	ed4	39.187%	88.401%	27,731	226m	852,043	25,749
diff8	mq4	ed4	39.162%	88.400%	27,731	319m	852,043	25,701
diff8	mq3	ed4	39.162%	88.400%	27,731	319m	852,043	25,701
diff8	mq2	ed4	39.162%	88.400%	27,731	319m	852,043	25,701
diff8	mq1	ed4	39.162%	88.400%	27,731	319m	852,043	25,701
diff6	mq4	ed4	39.129%	88.299%	27,730	272m	852,040	26,931
diff6	mq3	ed4	39.129%	88.299%	27,730	272m	852,040	26,931
diff6	mq2	ed4	39.129%	88.299%	27,730	272m	852,040	26,931
diff6	mq1	ed4	39.129%	88.299%	27,730	271m	852,040	26,931
diff5	mq5	ed4	39.109%	88.454%	27,730	166m	852,040	33,666
diff3	mq5	ed4	39.109%	88.454%	27,730	216m	852,040	33,666
diff5	mq9	ed4	39.091%	88.436%	27,730	165m	852,040	30,134
diff5	mq8	ed4	39.091%	88.436%	27,730	165m	852,040	30,134
diff5	mq7	ed4	39.091%	88.436%	27,730	166m	852,040	30,134

Diff	Match Quality	Edit Dist	PR-AUC	ROC-AUC	Num Revs	Run Time	Total Triangles	Bad Triangles
diff5	mq6	ed4	39.091%	88.436%	27,730	165m	852,040	30,134
diff3	mq9	ed4	39.091%	88.436%	27,730	214m	852,040	30,134
diff3	mq8	ed4	39.091%	88.436%	27,730	214m	852,040	30,134
diff3	mq7	ed4	39.091%	88.436%	27,730	215m	852,040	30,134
diff3	mq6	ed4	39.091%	88.436%	27,730	215m	852,040	30,134
diff6	mq9	ed4	39.059%	88.330%	27,730	478m	852,040	20,077
diff6	mq8	ed4	39.059%	88.330%	27,730	476m	852,040	20,077
diff6	mq7	ed4	39.059%	88.330%	27,730	488m	852,040	20,077
diff6	mq6	ed4	39.059%	88.330%	27,730	488m	852,040	20,077
diff5	mq4	ed4	39.022%	88.398%	27,730	152m	852,040	33,768
diff5	mq3	ed4	39.022%	88.398%	27,730	152m	852,040	33,768
diff5	mq2	ed4	39.022%	88.398%	27,730	152m	852,040	33,768
diff5	mq1	ed4	39.022%	88.398%	27,730	152m	852,040	33,768
diff3	mq4	ed4	39.010%	88.405%	27,730	202m	852,040	33,614
diff3	mq3	ed4	39.010%	88.405%	27,730	202m	852,040	33,614
diff3	mq2	ed4	39.010%	88.405%	27,730	202m	852,040	33,614
diff3	mq1	ed4	39.010%	88.405%	27,730	202m	852,040	33,614
diff1	mq9	ed4	38.790%	88.209%	27,731	96m	852,043	37,004
diff1	mq8	ed4	38.790%	88.209%	27,731	96m	852,043	37,004
diff1	mq7	ed4	38.790%	88.209%	27,731	96m	852,043	37,004
diff1	mq6	ed4	38.790%	88.209%	27,731	96m	852,043	37,004
diff1	mq5	ed4	38.783%	88.198%	27,731	96m	852,043	38,763
diff1	mq4	ed4	38.728%	88.192%	27,731	95m	852,043	37,424
diff1	mq3	ed4	38.728%	88.192%	27,731	95m	852,043	37,424
diff1	mq2	ed4	38.728%	88.192%	27,731	95m	852,043	37,424
diff1	mq1	ed4	38.728%	88.192%	27,731	95m	852,043	37,424
diff2	mq4	ed4	38.670%	88.280%	27,730	95m	852,040	56,968

Diff	Match Quality	Edit Dist	PR-AUC	ROC-AUC	Num Revs	Run Time	Total Triangles	Bad Triangles
diff2	mq3	ed4	38.670%	88.280%	27,730	95m	852,040	56,968
diff2	mq2	ed4	38.670%	88.280%	27,730	95m	852,040	56,968
diff2	mq1	ed4	38.670%	88.280%	27,730	95m	852,040	56,968
diff2	mq9	ed4	38.664%	88.263%	27,730	96m	852,040	54,214
diff2	mq8	ed4	38.664%	88.263%	27,730	96m	852,040	54,214
diff2	mq7	ed4	38.664%	88.263%	27,730	96m	852,040	54,214
diff2	mq6	ed4	38.664%	88.263%	27,730	96m	852,040	54,214
diff2	mq5	ed4	38.659%	88.225%	27,730	96m	852,040	54,354
diff6	mq5	ed4	38.559%	88.123%	27,730	492m	852,040	26,883
diff7	mq9	ed4	37.785%	88.077%	27,730	253m	852,040	33,481
diff7	mq8	ed4	37.785%	88.077%	27,730	252m	852,040	33,481
diff7	mq7	ed4	37.785%	88.077%	27,730	258m	852,040	33,481
diff7	mq6	ed4	37.785%	88.077%	27,730	258m	852,040	33,481
diff7	mq4	ed4	37.652%	88.026%	27,730	156m	852,040	40,767
diff7	mq3	ed4	37.652%	88.026%	27,730	156m	852,040	40,767
diff7	mq2	ed4	37.652%	88.026%	27,730	155m	852,040	40,767
diff7	mq1	ed4	37.652%	88.026%	27,730	156m	852,040	40,767
diff7	mq5	ed4	37.289%	87.920%	27,730	260m	852,040	41,985
diff7	mq9	ed2	30.210%	85.176%	28,448	252m	874,143	0
diff7	mq8	ed2	30.210%	85.176%	28,448	252m	874,143	0
diff7	mq7	ed2	30.210%	85.176%	28,448	258m	874,143	0
diff7	mq6	ed2	30.210%	85.176%	28,448	258m	874,143	0
diff7	mq4	ed2	30.193%	85.182%	28,448	156m	874,143	0
diff7	mq3	ed2	30.193%	85.182%	28,448	156m	874,143	0
diff7	mq2	ed2	30.193%	85.182%	28,448	155m	874,143	0
diff7	mq1	ed2	30.193%	85.182%	28,448	155m	874,143	0
diff7	mq5	ed2	30.146%	85.168%	28,448	260m	874,143	0

Diff	Match Quality	Edit Dist	PR-AUC	ROC-AUC	Num Revs	Run Time	Total Triangles	Bad Triangles
diff5	mq5	ed2	30.034%	85.309%	28,448	166m	874,143	0
diff3	mq5	ed2	30.034%	85.309%	28,448	216m	874,143	0
diff5	mq9	ed2	30.020%	85.281%	28,448	165m	874,143	0
diff5	mq8	ed2	30.020%	85.281%	28,448	165m	874,143	0
diff5	mq7	ed2	30.020%	85.281%	28,448	166m	874,143	0
diff5	mq6	ed2	30.020%	85.281%	28,448	166m	874,143	0
diff3	mq9	ed2	30.020%	85.281%	28,448	215m	874,143	0
diff3	mq8	ed2	30.020%	85.281%	28,448	215m	874,143	0
diff3	mq7	ed2	30.020%	85.281%	28,448	215m	874,143	0
diff3	mq6	ed2	30.020%	85.281%	28,448	215m	874,143	0
diff6	mq5	ed2	30.015%	85.206%	28,448	493m	874,143	0
diff3	mq4	ed2	30.010%	85.269%	28,448	202m	874,143	0
diff3	mq3	ed2	30.010%	85.269%	28,448	202m	874,143	0
diff3	mq2	ed2	30.010%	85.269%	28,448	202m	874,143	0
diff3	mq1	ed2	30.010%	85.269%	28,448	202m	874,143	0
diff5	mq4	ed2	30.008%	85.268%	28,448	152m	874,143	0
diff5	mq3	ed2	30.008%	85.268%	28,448	152m	874,143	0
diff5	mq2	ed2	30.008%	85.268%	28,448	152m	874,143	0
diff5	mq1	ed2	30.008%	85.268%	28,448	152m	874,143	0
diff2	mq9	ed2	29.986%	85.283%	28,448	96m	874,143	0
diff2	mq8	ed2	29.986%	85.283%	28,448	96m	874,143	0
diff2	mq7	ed2	29.986%	85.283%	28,448	96m	874,143	0
diff2	mq6	ed2	29.986%	85.283%	28,448	96m	874,143	0
diff6	mq9	ed2	29.983%	85.136%	28,448	476m	874,143	0
diff6	mq8	ed2	29.983%	85.136%	28,448	476m	874,143	0
diff6	mq7	ed2	29.983%	85.136%	28,448	488m	874,143	0
diff6	mq6	ed2	29.983%	85.136%	28,448	487m	874,143	0

Diff	Match Quality	Edit Dist	PR-AUC	ROC-AUC	Num Revs	Run Time	Total Triangles	Bad Triangles
diff6	mq4	ed2	29.963%	85.121%	28,448	272m	874,143	0
diff6	mq3	ed2	29.963%	85.121%	28,448	272m	874,143	0
diff6	mq2	ed2	29.963%	85.121%	28,448	272m	874,143	0
diff6	mq1	ed2	29.963%	85.121%	28,448	272m	874,143	0
diff2	mq5	ed2	29.962%	85.286%	28,448	96m	874,143	0
diff2	mq4	ed2	29.962%	85.270%	28,448	95m	874,143	0
diff2	mq3	ed2	29.962%	85.270%	28,448	95m	874,143	0
diff2	mq2	ed2	29.962%	85.270%	28,448	95m	874,143	0
diff2	mq1	ed2	29.962%	85.270%	28,448	95m	874,143	0
diff4	mq4	ed2	29.897%	85.250%	28,448	226m	874,143	0
diff4	mq3	ed2	29.897%	85.250%	28,448	227m	874,143	0
diff4	mq2	ed2	29.897%	85.250%	28,448	226m	874,143	0
diff4	mq1	ed2	29.897%	85.250%	28,448	226m	874,143	0
diff8	mq5	ed2	29.895%	85.273%	28,448	348m	874,143	0
diff4	mq5	ed2	29.895%	85.273%	28,448	255m	874,143	0
diff8	mq9	ed2	29.893%	85.249%	28,448	346m	874,143	0
diff8	mq8	ed2	29.893%	85.249%	28,448	346m	874,143	0
diff8	mq7	ed2	29.893%	85.249%	28,448	347m	874,143	0
diff8	mq6	ed2	29.893%	85.249%	28,448	347m	874,143	0
diff4	mq9	ed2	29.893%	85.249%	28,448	251m	874,143	0
diff4	mq8	ed2	29.893%	85.249%	28,448	251m	874,143	0
diff4	mq7	ed2	29.893%	85.249%	28,448	253m	874,143	0
diff4	mq6	ed2	29.893%	85.249%	28,448	253m	874,143	0
diff8	mq4	ed2	29.892%	85.247%	28,448	319m	874,143	0
diff8	mq3	ed2	29.892%	85.247%	28,448	320m	874,143	0
diff8	mq2	ed2	29.892%	85.247%	28,448	319m	874,143	0
diff8	mq1	ed2	29.892%	85.247%	28,448	319m	874,143	0

Diff	Match Quality	Edit Dist	PR-AUC	ROC-AUC	Num Revs	Run Time	Total Triangles	Bad Triangles
diff1	mq9	ed2	29.847%	85.222%	28,448	96m	874,143	0
diff1	mq8	ed2	29.847%	85.222%	28,448	96m	874,143	0
diff1	mq7	ed2	29.847%	85.222%	28,448	96m	874,143	0
diff1	mq6	ed2	29.847%	85.222%	28,448	96m	874,143	0
diff1	mq5	ed2	29.836%	85.242%	28,448	96m	874,143	0
diff1	mq4	ed2	29.803%	85.201%	28,448	95m	874,143	0
diff1	mq3	ed2	29.803%	85.201%	28,448	95m	874,143	0
diff1	mq2	ed2	29.803%	85.201%	28,448	95m	874,143	0
diff1	mq1	ed2	29.803%	85.201%	28,448	95m	874,143	0

Table F.1: Comparison of edit longevity performance, sorted by PR-AUC.

Bibliography

[1] B. Thomas Adler, Krishnendu Chatterjee, Luca de Alfaro, Marco Faella, Ian Pye, and Vishwanath Raman. Assigning trust to Wikipedia content. In *Proceedings of the 4th International Symposium on Wikis*, WikiSym 2008. ACM Press, 2008.

[2] B. Thomas Adler and Luca de Alfaro. A content-driven reputation system for the Wikipedia. In *Proceedings of the 16th International World Wide Web Conference*, WWW 2007. ACM Press, 2007.

[3] B. Thomas Adler, Luca de Alfaro, Santiago M. Mola-Velasco, Paolo Rosso, and Andrew G. West. Wikipedia vandalism detection: Combining natural language, metadata, and reputation features. In Alexander Gelbukh, editor, *CICLing 2011: Proceedings of the 12th International Conference on Intelligent Text Processing and Computational Linguistics*, volume 6609 of *Lecture Notes in Computer Science*, pages 277–288. Springer Berlin / Heidelberg, 2011.

[4] B. Thomas Adler, Luca de Alfaro, and Ian Pye. Detecting Wikipedia vandalism using WikiTrust. In Martin Braschler and Donna Harman, editors, *Notebook Papers of CLEF 2010 LABs and Workshops, 22-23 September, Padua, Italy*, September 2010.

[5] B. Thomas Adler, Luca de Alfaro, Ian Pye, and Vishwanath Raman. Measuring author contributions to the wikipedia. In *Proceedings of the 4th International Symposium on Wikis*, WikiSym 2008. ACM Press, 2008.

[6] Bo Adler, Luca de Alfaro, and Ian Pye. Redesigning scientific reputation. *The Scientist*, 24(9):30, September 2010.

[7] Denise Anthony, Sean W. Smith, and Tim Williamson. Explaining quality in Internet collective goods: Zealots and Good Samaritans in the case of Wikipedia. `http://web.mit.edu/iandeseminar/Papers/Fall2005/anthony.pdf`, November 2005. (Retrieved on 6-Mar-2011.).

[8] Amit Belani. Vandalism detection in Wikipedia: a bag-of-words classifier approach. *Computing Research Repository (CoRR)*, abs/1001.0700, 2010.

[9] Yochai Benkler. Coase's Penguin, or Linux, and The Nature of the Firm. *Yale Law Journal*, 112(3), December 2002.

[10] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, SP '96, pages 164–173. IEEE Computer Society, 1996.

[11] John Borland. See who's editing Wikipedia — Diebold, the CIA, a campaign, August 2007. `http://www.wired.com/politics/onlinerights/news/2007/08/wiki_tracker`.

[12] Moira Burke and Robert Kraut. Taking up the mop: identifying future Wikipedia administrators. In *Extended Abstracts Proceedings of the 2008 Conference on Human Factors in Computing Systems*, CHI 2008, pages 3441–3446. ACM, 2008.

[13] Randal C. Burns and Darrell D.E. Long. A linear time, constant space differencing algorithm. In *Proceedings of the 16th IEEE International Performance, Computing, and Communications Conference*, IPCCC 1997, pages 429–436. IEEE International, 1997.

[14] Brian Butler, Lee Sproull, Sara Kiesler, and Robert Kraut. Community effort in online groups: Who does the work and why. In Weisband and Atwater, editors, *Leadership at a Distance: Research in Technologically-supported Work*. Lawrence Erlbaum Associates, 2002.

[15] Jacobi Carter. ClueBot and vandalism on Wikipedia, 2008. [Online; accessed 2-Nov-2010].

[16] Jorge Cham. The origin of the theses, 2009. [Online; accessed 5-Mar-2012].

[17] Krishnendu Chatterjee, Luca de Alfaro, and Ian Pye. Robust content-driven reputation. In *Proceedings of the 1st ACM Workshop on Security and Artificial Intelligence*, AISec 2008. ACM Press, 2008.

[18] Si-Chi Chin, W. Nick Street, Padmini Srinivasan, and David Eichmann. Detecting Wikipedia vandalism with active learning and statistical language models. In *WICOW '10: Proceedings of the Fourth Workshop on Information Credibility on the Web*, Apr 2010.

[19] Clyde H. Coombs, Robyn M. Dawes, and Amos Tversky. *Mathematical Psychology: An Elementary Introduction*. Prentice-Hall, 1970.

[20] Graham Cormode and S. Muthukrishnan. The string edit distance matching problem with moves. *ACM Transactions on Algorithms*, 3(1):2:1–2:19, February 2007.

[21] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. J. Wiley & Sons, 1991.

[22] Tom Cross. Puppy smoothies: Improving the reliability of open, collaborative wikis. *First Monday*, 11(9), September 2006.

[23] Fred J. Damerau. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3), March 1964.

[24] Jesse Davis and Mark Goadrich. The relation between Precision-Recall and ROC curves. In *ICML'06: Proceedings of the 23rd International Conference on Machine Learning*, pages 233–240. ACM, 2006.

[25] Matthew Davis. Congress "made Wikipedia changes". *BBC News*, Feb. 9, 2006.

[26] Luca de Alfaro, Ashutosh Kulshreshtha, Ian Pye, and B. Thomas Adler. Reputation systems for open collaboration. *Communications of the ACM*, 54(8):81–87, August 2011.

[27] Chrysanthos Dellarocas. The digitization of word-of-mouth: Promises and challenges of online reputation systems. *Management Science*, 49(10):1407–1424, October 2003.

[28] W. Erwin Diewert and Alice O. Nakamura. Concepts and measures of productivity: An introduction. In Lipsey and Nakamura, editors, *Services Industries and the Knowledge Based Economy*. University of Calgary Press, 2005.

[29] Gregory Druck, Gerome Miklau, and Andrew McCallum. Learning to predict the quality of contributions to wikipedia. In *WikiAI'08: Proceedings of the Workshop on Wikipedia and Artificial Intelligence: An Evolving Synergy*, pages 7–12. AAAI Press, 2008.

[30] Michael D. Ekstrand and John T. Riedl. rv you're dumb: Identifying discarded work in Wiki article history. In *Proceedings of the 5th International Symposium on Wikis*, WikiSym 2009. ACM Press, 2009.

[31] William Emigh and Susan C. Herring. Collaborative authoring on the Web: A genre analysis of online encyclopedias. In *HICSS '05: Proceedings of the 38th Hawaii International Conference on System Sciences*, page 99a. IEEE Computer Society, 2005.

[32] Robert E. Park et al. Software size measurement: A framework for counting source statements. Technical Report CMU/SEI-92-TR-020, Carnegie Mellon University, September 1992.

[33] F. Randall Farmer and Bryce Glass. *Building Web Reputation Systems*. O'Reilly Media, Inc., 2010.

[34] Rudolph F. Flesch. A new readability yardstick. *Journal of Applied Psychology*, 32:221–233, 1948.

[35] Peter Kin-Fong Fong and Robert P. Biuk-Aghai. What did they do? deriving high-level edit histories in wikis. In *Proceedings of the 6th International Symposium on Wikis*, WikiSym 2010. ACM Press, 2010.

[36] Jim Giles. Internet encyclopedias go head to head. *Nature*, 438:900–901, 2005.

[37] Jennifer Ann Golbeck. *Computing and Applying Trust in Web-Based Social Networks*. PhD thesis, University of Maryland, 2005.

[38] Preston Gralla. U.S. senator: It's time to ban Wikipedia in schools, libraries. `http://blogs.computerworld.com/4598/u_s_senator_its_time_to_ban_wikipedia_in_schools_libraries`. [Online; accessed 15-Nov-2010].

[39] Virgil Griffith. Wikiscanner: List anonymous wikipedia edits from interesting organizations. `http://wikiscanner.virgil.gr/`.

[40] R. Guha, Ravi Kumar, Prabhakar Raghavan, and Andrew Tomkins. Propagation of trust and distrust. In *Proceedings of the 13th International Conference on World Wide Web*, WWW '04, pages 403–412. ACM, 2004.

[41] Robert Gunning. *The Technique of Clear Writing*. McGraw-Hill International Book Co., New York, NY, 1952.

[42] Dan Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1999.

[43] Aaron Halfaker, Aniket Kittur, Robert Kraut, and John Riedl. A judy of your peers: Quality, experience and ownership in wikipedia. In *Proceedings of the 5th International Symposium on Wikis*, WikiSym 2009. ACM Press, 2009.

[44] Aaron Halfaker, Aniket Kittur, and John Riedl. Don't bite the newbies: How reverts affect the quantity and quality of wikipedia work. In *Proceedings of the 7th International Symposium on Wikis*, WikiSym 2011. ACM Press, 2011.

[45] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I.H. Witten. The WEKA data mining software: An update. *SIGKDD Explorations*, 11(1), 2009.

[46] Martin Hickman and Genevieve Roberts. Wikipedia — separating fact from fiction. *The New Zealand Herald*, Feb. 13 2006.

[47] J. W. Hunt and M. D. McIlroy. An algorithm for differential file comparison. Computing Science Technical Report 41, Bell Laboratories, 1976.

[48] Kelly Y. Itakura and Charles L.A. Clarke. Using dynamic Markov compression to detect vandalism in the Wikipedia. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '09, pages 822–823. ACM Press, 2009.

[49] Karen Spärck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28(1):11–21, 1972.

[50] Patrick Juola. Authorship attribution. *Foundations and Trends in Information Retrieval*, 1(3):233–334, December 2006.

[51] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The eigentrust algorithm for reputation management in P2P networks. In *Proceedings of the 12th International Conference on World Wide Web*, WWW '03, pages 640–651. ACM, 2003.

[52] Raymond King. Contributor ranking system, 2007. White paper available from `http://trust.cse.ucsc.edu/Related_Work`.

[53] Daniel Kinzler. Personal communication, January 2011.

[54] Aniket Kittur, Ed Chi, Bryan A. Pendleton, Bongwon Suh, and Todd Mytkowicz. Power of the Few vs. Wisdom of the Crowd: Wikipedia and the rise of the Bourgeoisie. *Alt.CHI*, 2007.

[55] Jon .M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.

[56] Evan Lehmann. Rewriting history under the dome. *The Sun*, Jan. 27, 2006.

[57] Bo Leuf and Ward Cunningham. *The Wiki Way. Quick Collaboration on the Web*. Addison-Wesley, 2001.

[58] V. I. Levenshtein. Binary codes capable of correcting insertions and reversals. *Soviet Physics Doklady*, 10:707–710, 1966.

[59] Andrew Lih. Wikipedia as participatory journalism: Reliable sources? Metrics for evaluating collaborative media as a news resources. In *Proceedings of the 5th International Symposium on Online Journalism*, 2004.

[60] Benjamin Livshits and Thomas Zimmermann. Dynamine: Finding common error patterns by mining software revision histories. *SIGSOFT Software Engineering Notes*, 30(5):296–305, September 2005.

[61] Roy Lowrance and Robert A. Wagner. An extension of the string-to-string correction problem. *Journal of the Association for Computing Machinery*, 22(2):177–183, April 1975.

[62] Teun Lucassen and Jan Maarten Schraagen. Evaluating WikiTrust: A trust support tool for Wikipedia. *First Monday*, 16(5), May 2011.

[63] Owen Martin. Personal communication, January 2011.

[64] Deborah L. McGuinness, Honglei Zeng, Paulo Pinheiro da Silva, Li Ding, Dhyanesh Narayanan, and Mayukh Bhaowal. Investigation into trust for collaborative information repositories: A Wikipedia case study. In *Proceedings of the WWW '06 Workshop on Models of Trust for the Web*, MTW '06, 2006.

[65] Brian Mingus, Trevor Pincock, and Laura Rassbach. Using natural language processing to determine the quality of Wikipedia articles. In *Wikimania, Taipei, Taiwan*, 2007. `http://wikimania2007.wikimedia.org/wiki/Proceedings:BM1`.

[66] Santiago M. Mola Velasco. Wikipedia vandalism detection through machine learning: Feature review and new proposals. In Martin Braschler and Donna Harman, editors, *Notebook Papers of CLEF 2010 LABs and Workshops, 22-23 September, Padua, Italy*, September 2010.

[67] Santiago M. Mola-Velasco. Wikipedia vandalism detection. In *WWW 2011: Proceedings of the 20th International World Wide Web Conference*, pages 391–396. ACM Press, 2011.

[68] Eugene W. Myers. An O(ND) difference algorithm and its variations. *Algorithmica*, 1(2):251–266, 1986.

[69] Theodor Holm Nelson. *Literary Machines*. Mindful Press, 1981.

[70] Christine M. Neuwirth, Ravinder Chandhok, David S. Kaufer, Paul Erion, James Morris, and Dale Miller. Flexible diff-ing in a collaborative writing system. In *Proceedings of the 1992 ACM Conference on Computer-Supported Cooperative Work*, CSCW 1992, pages 147–154. ACM, 1992.

[71] Yuki Noguchi. Palin's Wikipedia entry gets overhaul. *All Things Considered*, Aug. 29, 2008.

[72] Wolfgang Obst. Delta technique and string-to-string correction. In *ESEC 1987: Proceedings of the 1st European Software Engineering Conference*, volume 289 of *Lecture Notes in Computer Science*, pages 64–68. Springer, 1987.

[73] Lynn Olanoff. School officials unite in banning Wikipedia. *The Seattle Times*, Nov. 21 2007.

[74] Felipe Ortega and Jesus M. Gonzalez-Barahona. Quantitative analysis of the Wikipedia community of users. In *Proceedings of the 3rd International Symposium on Wikis*, WikiSym 2007, pages 75–86, New York, NY, USA, 2007. ACM.

[75] Felipe Ortega, Jesus M. Gonzalez-Barahona, and Gregorio Robles. On the inequality of contributions to wikipedia. In *Proceedings of the 41st Annual Hawaii International Conference on System Sciences*, HICSS 2008. IEEE Computer Society, 2008.

[76] José Felipe Ortega Soto. *Wikipedia: A quantitiative analysis*. PhD thesis, Universidad Rey Juan Carlos, Madrid, Spain, 2009.

[77] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.

[78] Martin Potthast. Crowdsourcing a Wikipedia vandalism corpus. In *Proceedings of the 33rd International ACM SIGIR Conference*, SIGIR 2010, pages 789–790. ACM Press, Jul 2010.

[79] Martin Potthast, Benno Stein, and Robert Gerling. Automatic vandalism detection in Wikipedia. In *Proceedings of the 30th European Conference on IR Research (ECIR '08)*, volume 4956 of *LNCS*, pages 663–668. Springer-Verlag, 2008.

[80] Martin Potthast, Benno Stein, and Teresa Holfeld. Overview of the 1st International Competition on Wikipedia Vandalism Detection. In Martin Braschler and Donna Harman, editors, *Notebook Papers of CLEF 2010 LABs and Workshops, 22-23 September, Padua, Italy*, September 2010.

[81] Reid Priedhorsky, Jilin Chen, Shyong K. Lam, Katherine Panciera, Loren Terveen, and John Riedl. Creating, destroying, and restoring value in Wikipedia. In *Group'07: Proceedings of the International Conference on Supporting Group Work*, 2007.

[82] Evan Prodromou. personal communication, 2007.

[83] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2007. ISBN 3-900051-07-0.

[84] Joseph Reagle. Wikipedia as an open content community. `http://reagle.org/joseph/2004/behav/wikipedia.html`, 2004. (Retrieved on 6-Mar-2011.).

[85] Christoph Reichenberger. Delta storage for arbitrary non-text files. In *Proceedings of the 3rd International Workshop on Software Configuration Management*, pages 144–152. ACM, 1991.

[86] Paul Resnick, Richard Zeckhauser, Eric Friedman, and Ko Kiwabara. Reputation systems. *Communications of the ACM*, 43(12):45–48, 2000.

[87] Mikalai Sabel. Structuring wiki revision history. In *Proceedings of the 3rd International Symposium on Wikis*, WikiSym 2007. ACM Press, 2007.

[88] P. A. Samuelson. A note on the pure theory of consumer's behavior. *Economica*, 25(17):61–71, 1938.

[89] David Sankoff and Joseph B. Kruskal, editors. *Time Warps, String Edits, and Macromolecules: the Theory and Practice of Sequence Comparison*. CSLI Publications, 1999.

[90] Stacy Schiff. Know it all: Can Wikipedia conquer expertise? *The New Yorker*, Jul. 31, 2006.

[91] Herman P. Schultz. Software management metrics. Technical Report AD-A196 916, MITRE, May 1988.

[92] Katharine Q. Seelye. A little sleuthing unmasks writer of Wikipedia prank. *The New York Times*, Dec. 11, 2005.

[93] Katharine Q. Seelye. Snared in the web of a Wikipedia liar. *The New York Times*, Dec. 4, 2005.

[94] John Seigenthaler. A false wikipedia 'biography'. *USA Today*, Nov. 29 2005.

[95] Koen Smets, Bart Goethals, and Brigitte Verdonk. Automatic vandalism detection in Wikipedia: Towards a machine learning approach. In *WikiAI'08: Proceedings of the Workshop on Wikipedia and Artificial Intelligence: An Evolving Synergy*, pages 43–48. AAAI Press, 2008.

[96] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[97] BBC Staff. Fake professor in Wikipedia storm. *BBC News*, Mar. 6, 2007.

[98] Klauss Stein and Claudia Hess. Does it matter who contributes: a study on featured articles in the german wikipedia. In *Proceedings of the 18th conference on Hypertext and hypermedia*, HT '07, pages 171–174, New York, NY, USA, 2007. ACM.

[99] Randall Stross. Anonymous Source Is Not the Same as Open Source. *New York Times*, March 2006. `http://www.nytimes.com/2006/03/12/business/yourmoney/12digi.html`.

[100] Bongwon Suh, Ed H. Chi, Aniket Kittur, and Bryan A. Pendleton. Lifting the veil: improving accountability and social transparency in Wikipedia with wikidashboard. In *Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, CHI '08, pages 1037–1040, New York, NY, USA, 2008. ACM.

[101] James Surowiecki. *The Wisdom of Crowds: Why the Many are Smarter Than the Few and How Collective Wisdom Shapes Business, Economies, Societies and Nations*. Doubleday, 2004.

[102] Aaron Swartz. Who writes Wikipedia? `http://www.aaronsw.com/weblog/whowriteswikipedia`, September 2006. (Retrieved on 6-Mar-2011.).

[103] Chris Taylor. Why commercial Wikis don't work. `http://money.cnn.com/2007/02/21/magazines/business2/walledgardens.biz2/index.htm`, February 2007. (Retrieved on 9-May-2008.).

[104] Geoff Tennant. *SIX SIGMA: SPC and TQM in Manufacturing and Services*. Ashgate Publishing, 2001.

[105] Walter F. Tichy. The string-to-string correction problem with block move. *ACM Transactions on Computer Systems*, 2(4):309–321, 1984.

[106] Hal R. Varian. Revealed preference. In M. Szenberg, L. Ramrattan, and A. Gottesman, editors, *Samuelsonian Economics and the 21st Century*. Oxford University Press, 2006.

[107] Fernanda B. Viégas, Martin Wattenberg, and Kushal Dave. Studying cooperation and conflict between authors with history flow visualizations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 575–582. ACM Press, 2004.

[108] Jakob Voß. Measuring wikipedia. In *Proceedings of the 10th International Conference of the ISSI*, 2005.

[109] Robert A. Wagner. On the complexity of the extended string-to-string correction problem. In *Proceedings of the 7th Annual ACM Symposium on Theory of Computing*, STOC 1975. ACM, 1975.

[110] Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *Journal of the Association for Computing Machinery*, 21(1):168–173, January 1974.

[111] Jimmy Wales. Wikipedia, emergence, and the wisdom of crowds. `http://lists.wikimedia.org/pipermail/wikipedia-l/2005-May/021764.html`, May 2005. (Retrieved 6-Mar-2011.).

[112] Andrew G. West, Sampath Kannan, and Insup Lee. Detecting Wikipedia vandalism via spatio-temporal analysis of revision metadata. In *EUROSEC'10: Proceedings of the Third European Workshop on System Security*, pages 22–28, 2010.

[113] Wikipedia. Politics of denmark — Wikipedia, the free encyclopedia, September 2006. `http://en.wikipedia.org/w/index.php?title=Politics_of_Denmark&oldid=77625823`.

[114] Wikipedia. Politics of denmark — Wikipedia, the free encyclopedia, September 2006. `http://en.wikipedia.org/w/index.php?title=Politics_of_Denmark&oldid=77692452`.

[115] Wikipedia. Information wants to be free — Wikipedia, the free encyclopedia, 2008. [Online; accessed 21-Feb-2011].

[116] Wikipedia. Nupedia — Wikipedia, the free encyclopedia, 2008. [Online; accessed 21-Feb-2011].

[117] Wikipedia. Wikipedia: Recent changes patrol — Wikipedia, the free encyclopedia, 2008. [Online; accessed 21-Feb-2011].

[118] Wikipedia. User:antivandalbot — Wikipedia, the free encyclopedia, 2010. [Online; accessed 2-Nov-2010].

[119] Wikipedia. User:cluebot — Wikipedia, the free encyclopedia, 2010. [Online; accessed 2-Nov-2010].

[120] Wikipedia. User:martinbot — Wikipedia, the free encyclopedia, 2010. [Online; accessed 2-Nov-2010].

[121] Wikipedia. Software quality — Wikipedia, the free encyclopedia, 2011. [Online; accessed 13-Mar-2011].

[122] Wikipedia. Triangle inequality — Wikipedia, the free encyclopedia, 2011. [Online; accessed 27-Apr-2011].

[123] Wikipedia. Web 2.0 — Wikipedia, the free encyclopedia, 2011. [Online; accessed 5-Mar-2011].

[124] Wikipedia. Wikipedia:stiki — Wikipedia, the free encyclopedia, 2011. [Online; accessed 8-Oct-2011].

[125] Dennis Wilkinson and Bernardo Huberman. Cooperation and quality in Wikipedia. In *Proceedings of the 3rd International Symposium on Wikis*, WikiSym 2007, pages 157–164. ACM Press, 2007.

[126] Honglei Zeng, Maher A. Alhossaini, Richard Fikes, and Deborah L. McGuinness. Mining revision history to assess trustworthiness of article fragments. In *Proceedings of the 2nd International Conference on Collaborative Computing: Networking, Applications, and Worksharing*, COLLABORATECOM, 2006.

[127] Honglei Zeng, Maher A. Alhoussaini, Li Ding, Richard Fikes, and Deborah L. McGuinness. Computing trust from revision history. In *Proceedings of the 2006 International Conference on Privacy, Security and Trust*, 2006.