University of London
Imperial College of Science, Technology and Medicine
Department of Computing

Automatically defining collaborative share through the automatic, naïve string analysis of Wikipedia article revision histories

William Marsey

Submitted in part fulfilment of the requirements for the degree of
Master of Science in Computing Science of Imperial College, 5 September 2014

Abstract

In this project, we analyse Wikipedia revision histories, with a view to automatically deriving an collaborative share for each of the article's editors. We use automatic, semantically-naïve string analysis as a basis for discussing the value of each contribution in an article's history. We analyse the edit distance between texts in various ways, and, in particular, try to characterise the each step of an article's path as connected to, completely contextualised by, every other in that history.

As a Wikipedia article is something that evolves organically out of the actions of many competing agents, then we may measure the success of a contribution – the smallest unit of which is a character – to be it's survival rate. Other studies have remarked upon this also. However, previous studies that endeavour to characterise or measure this survival rate concentrate on identifying binary do/undo interaction between edits, which – as we'll see – is a common occurrence, but we feel is inadequate in terms of really understanding the constant, nuanced to-and-fro of mass on-line collaboration. We offer a solution in our 'trajectory' technique.

We also consider the text as composed of numerous different species of text, dividing it using Wikipedia's native mark-up convention, 'wikimarkup', and look to find correlations between text species and survival rate.

We find that, on Wikipedia in particular, that qualitative analysis of text, at least characterised by species in this way, does not correlate well to survival.

Though this observation may render a fair amount of the text-analysis work obsolete in the case of analysing Wikipedia in particular, the produced software is robust and malleable, and the procedures developed and implemented here would lend themselves well to analysing the interaction-driven analytical paradigm aforementioned. As it stands, this work would adapt well to analysis of more stable forms of on-line contribution, such as source control models.

"This page documents . . . a generally accepted standard that editors should follow, though it should be treated with common sense, and occasional exceptions may apply. Changes made to it should reflect consensus; when in doubt, discuss your idea on the talk page."

"Once an article reaches the A-Class, it is considered "complete", although edits will continue to be made."

from *Wikipedia:Version 1.0 Editorial Team/Assessment, en.wikipedia.org*

Contents

List of Algorithms

List of Figures

Chapter 1

Introduction

1.1 Objectives

Our principal aim was to design a series of procedures for distributing a 'share' of a collaborative article to each of that article's contributors. Traditionally these shares are negotiated on a case-by-case basis, through individual negotations, or according to balances struck previously. We discuss this in further detail on page 3.

Our investigation instead looks to automatic share distribution in the environment of many-user on-line collaboration; how may we automatically define a individual user's stake in a collaborative work?

Wikipedia is the natural choice for the study, though defining user 'ownership' in this particular context seems quite unnatural. The website that bills itself as the 'free encyclopaedia that anyone can edit' can hardly have any obligation to its users to define a sense of individual ownership of the article, whole or otherwise.[76] However, as an intellectual exercise, the question is very relevant, with on-line platforms increasing in popularity in education,[51] and the workplace,[16] and generally becoming a 'necessity' in day-to-day life.[6]

Thankfully, the technological platforms that allow projects such as Wikipedia to grow also facilitate their study. Using the technology, we may may examine how to describe the nature of an individual contribution amongst many. Wikipedia is, after all, one of the most popular sites on the internet, and certainly one the most popular reference site on the internet (see footnote page 8).

On a more practical level, Wikipedia represents a free, flexible and (fairly)[1] complete data set with which to experiment analysing incremental, oblique and convoluted text change over long periods of time. From this base work we can begin to approach analysing other data sets, such as source-control platforms like Github.

Another benefit of analysing Wikipedia is that interactions between users are mostly mediated by the Wikipedia software, and in exactly the same way the articles are, allowing us to examine a relatively wide range of site properties using a similar set of tools.

[1] The API proved to be quite unreliable. This is discussed on page 11.

For this project, we produce a system for automatically downloading an article's history, along with various methods with which to process and describe that data in a meaningful way. We use a Levenshtein edit-distance calculating algorithm in order to examine the magnitude of change between edits, as well as to examine the context of those edits.

For each revision we calculate a set of integers and real numbers that represent a multi-dimension characterisation of the change that edit caused in the article. We provide other means to express the degree to which that edit was significant in terms of the history of that article thus far.

1.2 Contributions

The major contribution here is a new way of visualising the article's history, and a technique for translating that into a meaningful, real-numbered value for each revision.

We also discuss using this technique describe various other contexts that come to bear upon journey of each article. Our main technique involves calculating the similarity of every revision to the final version. However, using the same software we may also track the history of the talk page for each article alongside the article itself, as well as test for the existence of 'arbitration requests' and other artefacts of Wikipedian bureaucracy that may shape an article's growth.

The software produced here automates the analytical procedures we outline in chapter 3, as well as providing a modular framework that may be modified for further study.

1.3 Originality

There has been a lot of studies tangential to the topics we discuss here – as we'll cover in chapter 2, Wikipedia is a hotbed of academic study in many disciplines – but much of the similar work to this works to different ends, and therefore a lot of the work here is original work.

The most closely related work is that of WikiTrust software. It was developed in the context of criticism's of Wikipedia's factual integrity, and aimed to rate the the editors of each article, transforming each article into a kind of heat map or trustability, highlighting words from more trusted editors more intensely than others.[4] This study has been evaluated as ineffective, however.[28]

In this project we use wikipedia as a model for automatically discovering appropriate measures of collaborative share. We are less concerned about 'quality' in any arbitrary, external sense. Rather we are interested how we may define internal quality using the nature of the history itself. That said, a lot of the studies are very interesting in light of our work. One of the techniques used in WikiTrust is very important in our conception of 'trajectory'. We discuss this more on page 8.

We offer less judgements on quality in this project, and our aims are more general, but some of the techniques resurface here. In particular, we modify the WikiTrust method for identifying undone work to create a much more detailed flow of change.

This project was also undertaken for the MSc degree at Imperial in 2013. That project has been taken into account, but as it contained no real developments upon existing studies, has been expanded upon greatly here.

Chapter 2

Background Theory

2.1 Copyright

We first look to existing copyright legislation on joint authorship. In particularly, we look to Intellectual property law – laws that govern 'creations of the mind'. It may be argued that Wikipedia writers are engaged in 'creations of the mind', but are rather analogous to copy writers. In any case the rules are the same – the content, and the construction of the content, is the part that matters, not the ideas themselves.[40]

We keep in mind that there is no world standard legislature regarding intellectual property, though Wikipedia is an international platform. (They are governed by United States copyright law, but endeavour to respect the copyright law of all countries, even if these do not have official copyright relations with the United States.[66])

Firstly, we review the general definitions of a joint work. According to American legislature:

> When two or more authors prepare a work with the intent to combine their contributions into inseparable or interdependent parts, the work is considered joint work and the authors are considered joint copyright owners.[63]

We can apply this confidently to Wikipedia. Editors of course know their efforts will be combined. The UK legislature is similar, defining a joint work thus:

> Where two or more people have created a single work protected by copyright and the contribution of each author is not distinct from that of the other(s).[39]

Though, joint authorship has no direct implications with regard to licensing. As far as most law is concerned, in joint-authored works, terms of usage, and therefore distribution of funds, must be agreed separately in each case:

> ownership of copyright can be transferred, so where something is produced that has involved contributions from more than one person, it would be possible for copyright in all the material to be owned by one person as a result of appropriate transfers. Indeed, collaborators can agree in advance that copyright in what is to be produced should be owned by a single person or body.

...

> However, alternative solutions that might be equally helpful could involve all parties agreeing licensing arrangements in advance.[39]

In the former case described above, the single 'owner' will sometimes be assigned the rights of all parties on agreement that they divvy up any profits according to the contract of that assignment. These profits are the parties' 'royalty rates'. In the latter case, the division of these proceeds by a pre-agreed royalty rate is a given.

How these royalty rates are calculated, however, is left to the discretion of the individuals involved. Industry literature defines the different methods for deciding upon these different rates: comparison with previous similar deals done by others, alignment with industry or internal practice, and calculation. The first method doesn't necessary reflect differences in cases, the second can be difficult when some parties have limited bargaining power. The third can be the most rational, though is often highly complex.[53]

This latter-most method seems to be the most applicable model to address the current situation. However, current models that have the royalty share based upon financial investment are inappropriate.[53] To achieve our goal of dividing share automatically, we are faced with defining a different kind of economy.

Writers are most generally hired by the hour / day, or commissioned by word count.[38] We have no data about how long an editor may spend on an article, and when there is a high likelihood that an edit will be refined, or partially deleted or rewritten by another, word count seems to be fairly inappropriate also.

In the context of the organic, open-source development of a Wikipedia article (Wikipedia's answer to who is responsible for articles? "You are!"[72]), it seems most natural to pursue measures of how an edit may have positively affected the growth of an article — it's consensus.

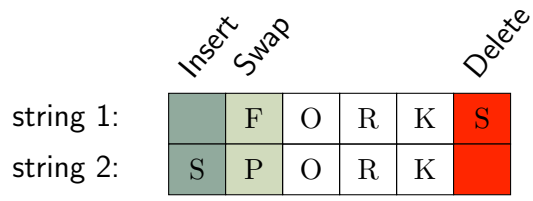
We can define a positive edit in two ways. A positive edit may be one that upholds the values of a good Wikipedia article, i.e. they are:

> written very well, contain factually accurate and verifiable information, are broad in coverage, neutral in point of view, stable, and illustrated, where possible, by relevant images with suitable copyright licenses.[77]

Or, they help the article along towards it's final version. In this we may approach the state above according to this heuristic:

> An article will automatically approach the 'golden' standard of Wikipedia.

According to this heuristic, if a majority of editors approach an article with the above standards in mind, any edits that do not meet these standards will be undone or changed so as to approach these standards. And the heuristic seems plausible. Indeed, Wikipedia even has it's own Wikipedia-local definition of the word consensus: "the primary way decisions are made on Wikipedia, and it is accepted as the best method to achieve *our goals*."[71]



forks → spork, edit distance: 3

Figure 2.1: An edit distance example using all three basic edit operations

for the function $\text{lev}_{a,b}(|a|, |b|)$:

$$\text{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } min(i,j) = 0 \\ \min \begin{cases} lev_{a,b}(i-1,j) + 1 \\ lev_{a,b}(i,j-1) + 1 \\ lev_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & else \end{cases}$$

when $a_i = b_j$, $1_{(a_i \neq b_j)} = 1$
when $a_i \neq b_j$, $1_{(a_i \neq b_j)} = 0$

Figure 2.2: The definition of Levenshtein edit distance.

According to these two techniques, we define an economy based on adherence to native Wikipedia standards of quality, and divide share based upon this.

2.2 Edit difference algorithms

To measure difference between different text revisions, we refer to edit distance. The edit distance between two texts, first defined by Vladimir Levenshtein in the 1960s,[25] can be defined as the minimum amount of insert, delete and substitutions operations needed to transform one text into another. We find a trivial illustration of the edit distance primitives in figure 2.1

Levenshtein's own characterisation of this distance is given in figure 2.2. It defines that the distance between two strings is characterised the minimum distance between three different pair-combinations of its sub-strings. A 'text-book' implementation of this algorithm can be represented by the pseudo-code in figure 1. We present the dynamic-programming-style algorithm here, and will generally be working with dynamic programming implementations throughout the study.

We can see that on comparing strings x and y, a $|x|$ by $|y|$ (length of x, length of y) table is created, and then filled with values. For this reason both the time and space complexity of the algorithm is $\theta(|x||y|)$.

Reducing the space needed for this computation is relatively easy, and can be done in a few different ways. One way is to simply disregard parts of the table already computed. We can see that, on each computation of $d[i,j]$, we require only a small part of the matrix: $d[i-1,j-1]$, $d[i-1,j]$ and $d[i,j-1]$. At any iteration i, where is great than 1, we may disregard rows $0 \ldots (i-2)$ inclusive. We eventually implement this version, implementing the algorithm 6, on page 22.

Algorithm 1 'Textbook' dynamic algorithm for computing Levenshtein distance

```
function ED(str1, str2)
    s1len ←length(str1)
    s2len ←length(str2)
    if s1len = 0 then
        return s2len
    end if
    if s2len = 0 then
        return s1len
    end if
    d ← [0 ... s1len][0 ... s2len]
    for i ← 1, s1len do
        d[i][0] ← i
    end for
    for j ← 0, s2len do
        d[0][j] ← j
    end for
    for j ← 1, s2len do
        for i ← 1, s1len do
            c ← 0 if str1[i − 1] ≠ str2[j − 1] else 1
            insert ← d[i − 1][j] + 1
            delete ← d[i, j − i] + 1
            keepswap ← d[i − 1][j − 1] + c
            d[i][j] ← min(insert, delete, keepswap)
        end for
    end for
    return d[s1en][s2len]
end function
```

There are more complicated techniques that allow us to also disregard unnecessary computation — a few implementations employ strategies that allow them to trace the table space diagonally, rather than iteratively, achieving a time complexities as low as $O(ed(x,y)^2)$ (though they sacrifice some accuracy).[9] Others harnesses the speed of bit operations to achieve a time complexity of $O(nm/w)$ or $O(nmlog\Sigma/w)$ time where w the bit-word size of the machine, and Σ is the alphabet size.[36][20]

In this project, however, it was suffice to simply reduce the space needed for the computation, as the texts were relatively small, and speed not an issue. However, we were able to speed up the program somewhat by implementing some simple multiprocessing. Both these procedures, and their limitations, are discussed in more detail on page 18.

Optimal alignment

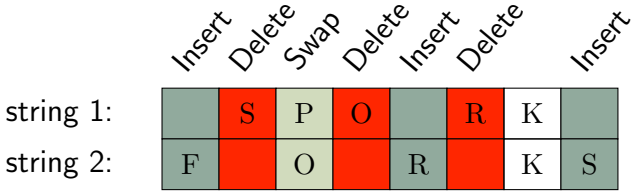
Another part of the problem of working out optimal edit distance is finding the 'optimal alignment' — the measures are closely related. We displace and arrange the characters of a string such that the set of operations to transform each character into its counterpart is minimal. For example, in figure 2.1, the alignment of the two strings "fork" and "spork" was:

s	p	o	r	k	-
-	f	o	r	k	s

However it could also conceivably have been:

s	p	o	r	k	-	or even	-	s	p	o	-	r	k	-
f	-	o	r	k	s		f	-	o	-	r	-	k	s

Here, the left-hand version results in an equivalent Levenshtein distance, but we can see how the distance for the right-hand example would be sub-optimal, requiring 7 edit operations.



spork → forks, edit distance: 7

Figure 2.3: An sub-optimal edit distance example

The Smith-Waterman algorithm calculates optimal alignment by populating two tables – one like that in the pseudocode above, and also as a table of directions.[57] These directions describe paths from one corner of the table space to the other; the shape of this path can define how to align the two strings.[58]

This path may also be read as an edit operation. An arrow at the position $[i, j]$ in the table defines edit operations for $x[i]$ and/or $y[j]$, as described in figure 2.4.

Our early implementations of the Levenshtein algorithm implemented various ways of computing this alignment data, including creating a separate matrix as here, and implementing a struct

$$\begin{Bmatrix} & & S & P & O & R & K \\ & 0 & 0 & 0 & 0 & 0 & 0 \\ F & 0 & \nwarrow & \nwarrow & \nwarrow & \nwarrow & \nwarrow \\ O & 0 & \uparrow & \nwarrow & \nwarrow & \downarrow & \leftarrow \\ R & 0 & \uparrow & \uparrow & \uparrow & \nwarrow & \leftarrow \\ K & 0 & \uparrow & \uparrow & \uparrow & \uparrow & \nwarrow \\ S & 0 & \nwarrow & \uparrow & \uparrow & \uparrow & \uparrow \end{Bmatrix}$$

(If the arrow path reaches an edge before the left-hand corner, we trace along that edge, reading each shift as an arrow in the direction of the trace.)

- $\nwarrow$ **at** $[i, j]$**, if** $x[i] \neq y[j]$
  Denotes a 'swap' between $x[i]$ and $y[j]$ (if $x[i] = y[j]$ then it denotes the lack of an operation).
- $\uparrow$ **at** $[i, j]$
  Denotes the deletion of $x[i]$
- $\leftarrow$ **at** $[i, j]$
  Denotes the insertion of $y[j]$

Figure 2.4: Diagram showing Smith-Waterman traceback path (in red) on the edit operation forks → spork

which would accrue this path data as it was passed along in the $min(a, b, c)$ function of the Levenshtein algorithm. However, these implementations were very slow, and we decided that evidencing alignment was not a priority.

We did, however, find that splitting the string caused some sub-optimal alignment, and discrepancies in plain levenshtein vs split levenshtein distance. We discuss this in detail on page 39.

2.3 Wikipedia

Wikipedia is a free, open-source, publicly-editable on-line knowledge-base. The software is runs upon, the PHP-based MediaWiki, is also open-source, powering countless other on-line encyclopaedias. The website is ranked 6^{th} globally in terms of website traffic, and is the highest-ranked reference website by far - most of the sites it shares the top spots with are commercial portals, search engines, shopping mega-sites, and social media websites.[1] Despite early scepticism (particularly concern over the inherent chaos in the system: "...edits, contributed in a predominantly undirected and haphazard fashion by ... unvetted volunteers."[79]), it is widely claimed to be a success, 'the best-developed attempt thus far of the enduring quest to gather all human knowledge in one place'[34].

That Wikipedia has become a hub of research in many fields is also self-evident to anyone who has searched for articles on the subject. Mesgari et al, just quoted, has prepared a very recent 'systematic review of scholarly research on the content of Wikipedia', which gives an overview of 110 articles on the subject — attesting to his observation that Wikipedia has been 'irresistible point of inquiry for researchers from various fields of knowledge'. It will be a useful touching stone

[1] According to 'Alexa', an website ranking company.[7] Though, this may be an underestimation. Alexa may well be biased towards English speakers and Internet Explorer users, underestimating Wikipedia.org's popularity, since 'two thirds of all Wikipedia articles are in languages other than English'[69]

for this study, finding 82 out of the 110 surveyed articles to concern Wikipedia quality. Some of these are also referenced here.

Other important general sources have been WikiLit,[67] AcaWiki[2] and WikiPapers[70], all of which are on-line repositories of academic research into Wikipedia and other Wikis, and from which many of the studies mentioned here were found.

Existing studies of Wikipedia revision history

Tangentially related studies fall into two major groups: studies of Wikipedia article quality and studies of edit behaviour. It is from the first group that we find the most pertinent work — it is also one of the most fruitful areas of research.

It is the metrics used to measure quality in these studies that are of most use to us here. We don't concern ourselves with the external quality rather how successful the article is in its own context of Wikipedia, but many studies have endeavoured to find out what kind of article content can be automatically recognised. We can also assume that editors may well appraise an article in a similar way to the passive, Wikipedia-innocent reader-agents these studies concern themselves with.

High numbers of Links, internal links, images and formulas have been found to indicate perceived quality,[29][32] and these are easy to identify using Wikipedia's mark-up language. Other useful metrics have been the age of the word,[11] the age and rate of change of the article in comparison to other articles,[81] and the recent activity of the article (an article undergoing a peak in edit changes may be 'unstable').[12] Another study of particular interest is that of Stvilia et al, which found metrics of article quality through factor analysis,[59] confirming much of the ideas already mentioned. A particularly interesting study found simply that word count maps well to success within Wikipedia, with longer articles receiving 'featured article' status.[8]

A landmark piece of work is the Wikitrust software.[5] Wikitrust was[2] a firefox plug-in, designed to highlight the words of a Wikipedia article with different colors, building upon the writing of Cross, 2006.[11] The gradations of these colors relate to levels of 'trust', and in turn on both the word's age, and the trust rating of the editor that contributed that word. A screen-shot can be seen in figure 2.5.

The program was reviewed as recently as 2011,[28] and it was found to be basically flawed, with users not really seeing the use for it. In fact, as a foil to the 'naïve reader' paradigm assumed by many of these studies, it was found that, having used Wikipedia before, readers already had personal methods for deciding whether to trust of not trust a Wikipedia article.

A few other studies present us with useful analyses of edit behaviour. These studies turn to Wikipedia in order to study user network interaction, and these analyses of conflict between editors show the possible reversion cases we will have to automatically recognise.

They reveal the high number of immediate 'undo'-type revisions, and also that malicious or unnecessary input may survive several versions before being undone. Some study these conflicts as a

[2]Defunct as of author's checks, Apr 2014

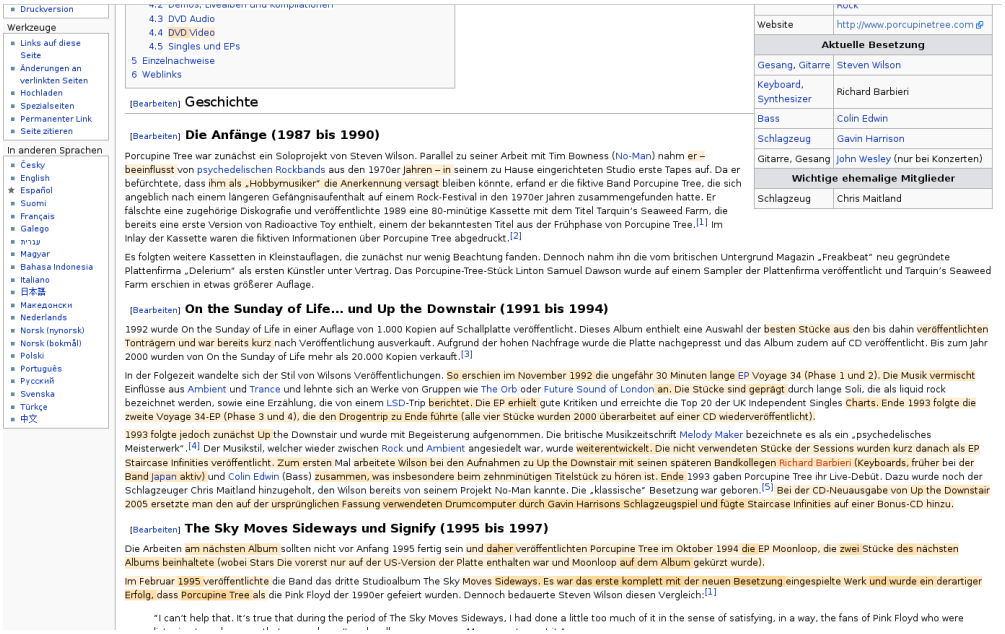


Figure 2.5: Wikitrust in action, 2011

characterisation of normal editing behaviour,[24][22][23][46] while others look to controversial articles,[21] or articles recently cited in the press.[27] We find from these same studies that articles lead by small groups of 'leaders' produce articles of better quality than those with a more homogeneous contribution group, that a small group of editors contribute to most of Wikipedia, and that conflict and bureaucracy (increasing over time) are the major limiting factors in the growth of an article.[61]

Chapter 3

Method

Our basic method for acquiring and analysing data can be broken down into three steps:

Firstly we send HTTP requests to the Wikipedia API, tracing history back from the most recent version to the least. We store the content of each revision, as well as data pertaining to the user, the time of the edit, it's size, and so forth. We store these in a PostgreSQL database.

Secondly we take the most recent revision, and compare it to all previous revisions in turn. We do this using a plain Levenshtein distance. It gives us an idea of how close the article was to its final version at each point in its history. We translate this into a real number for each revision which takes into account the change in time between each revision and the next, as well as the difference in their distance-from-final.

Thirdly we measure the Levenshtein distance between each child-parent revision pair, splitting the text into various different species, and measuring them separately. This results

The rest of this chapter is an elaboration of this three-step plan.

Data collection and storage

WikiMedia's API service provides simple access to a wiki data, features and meta-data over HTTP,[33] and for this project, provides the entirety of our data. In this section we explore the process of collecting Wikipedia data, the peculiarities of Wikipedia as a data source, and the algorithmic caveats necessary to deal with them.

Our basic request is simple: we send a HTTP request to a given wiki site's 'wiki/api.php' file, sending the query parameters a 'prop=revisions', 'rvprop=content', and 'titles=X|Y' to get those page's most recent revision contents. We can also prefetch a title using a random request (parameters 'list=random&rnlimit=1'), and in each case we can add the 'format=json' parameter, and quite easily parse the results. These simple techniques form the basis of our data collection – the more complex operations of the API caused various problems, discussed below.

To trace the history of a given page, we need only augment the aforementioned 'rvprop' argument to include 'ids' ('rvprop=content|ids') in order to discover the parent id and, from there, we may trace

the history backwards. The procedure defined in algorithm 2 demonstrates this clearly, showing the child-parent swap at line 22.

However, the API, even in these simple operations, is not totally reliable. The condition on the while loop (line 6), however, shows the first oddity with the Wikipedia histories. Most articles will terminate at their origin, showing parentid 0. Some, instead, enter a terminal cycle, with the oldest fetched revision giving its parent to be a much younger version of the same article. We found this to occur consistently with some articles, to imply that the problem is not just a temporary glitch in the delivery of the data.

Algorithm 2 Data fetching

```
    procedure FETCH(pageid)
        corrupt ← ∅
        visitedpages ← ∅
        revid ← 0
 5:     parentid ← wiki.getlatest(pageid)
        while revid ≠ 0 AND revid ∉ visitedpages do
            if revid is in the database then
                parentid ← database.getparentid(revid)
            else
10:             pagedata ← wiki.getpage(revid)
            end if
            if pagedata is corrupt then
                if corruptness is within recoverability bounds then
                    corruptpages ← corruptpages + (revid, parentid, domain)
15:             else
                    terminate fetch
                end if
            else
                database ← pagedata
20:         end if
            visitedpages ← visitedpages + (revid, domain)
            revid ← parentid
        end while
        for all (revision, parent, domain) ∈ corrupt do
25:         CorruptClean(revision, parent, domain)          ▷ See algorithm 3
        end for
        Mark pageid as complete in database
    end procedure
```

So, we keep a track of all the revisions we already know exist of that article, so that we may identify these cycles, and terminate the fetch loop early. In early tests, we find that these cycles occur exclusively amongst older articles (though not all old articles have the problem). For the purposes of this study, then, we must acknowledge that any 'complete' history of an article in our databases is in fact only the complete *discoverable*[1] history. We were unable to find an explanation for this problem on-line.

A second notable problem was the corrupt data regularly returned by the Wikipedia API. We test

[1]We may note here that, using our fetch algorithm, we have often managed to trace histories beyond the oldest listed on the site itself. That these older revisions are not available on the normal site implies a compromise, perhaps an early server problem.

for 'corruption' at line 12 in algorithm 2, storing a list of corrupt pages. Here we test for problems that do not effect the fetch overall, but would cause problems during the analysis, such as missing timestamps, or user data. In these cases 'circumnavigate' corrupt entries in the database, using the procedure in algorithm 3. The database changing pointers between children and parents so as to remove the corrupt revisions from the chain. We may then later trace a history of incorrupt entries using only these pointers.

In line 16 of algorithm 2 we define some corruptness of data to constitute a fetching failure, and terminate the fetch altogether. With our model, we only do this when parentid information is missing. Terminating the loop early, in practice, means either discarding all fetched data, or terminating the history at that point, depending on whether we have fetched enough revisions to satisfy our 'scrape minimum' limit.

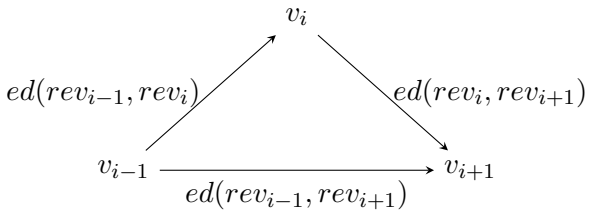
On exiting procedure mark a pageid as having a successful fetch in the database. This is useful for fetches that are interrupted unexpectedly, by hardware or connection problems. It allows us to clean or repair the database further down the line.

Algorithm 3 Corrupt pages

```
procedure CorruptClean(corruptrev, parent, domain)
    childrev ← database.getchildof(corruptrev)
    database.setparent(childrev, parent)
end procedure
```

Undone and partially undone operations

We consider three different ways of classifying an edit as valueless, or partially valueless. Two ways of classifying these kinds of edit are found in previous research, and are covered in figure 3.1, taken (modified) from the Wikitrust work by Adler, et al.[5].



case a) if $ed(rev_{i-1}, rev_i) < ed(rev_{i-1}, rev_i) + ed(rev_i, rev_{i+1})$,
then $ed(rev_{i-1}, rev_i)$ has been partially undone.

case b) if $ed(rev_{i-1}, rev_{i+1}) = 0$ $(rev_{i-1} = rev_{i+1})$,
then $ed(rev_{i-1}, rev_i)$ has been completely undone.

Figure 3.1: Diagram showing identification of a partially or completely undone edit

Figure 3.1 shows how we may identify undone or partially undone edits, when they are undone immediately, as laid out in Adler et al.[5] The triangle represents three consecutive revisions, and the arrows are the edit operations that transform one to another. By calculating the edit distance between the distance versions rev_{i-1} and rev_{i+1} we may characterise a longer history of revisions

than usual, and use that context in order to re-characterise the edits it encompasses. In the figure above, case a describes rev_i as a 'diversion'. If rev_{i-1} can be transformed into rev_{i+1} with less operations than the two edits that actually bridged that gap, then perhaps some of the edits in $rev_{i-1} \rightarrow rev_i$ were unnecessary, and undone by rev_{i+1}. In this case, we may punish the edit $rev_{i-1} \rightarrow rev_i$ (for the diversion), reward $rev_i \rightarrow rev_{i+1}$, or both. Case b is an extreme version of a — the texts rev_{i-1} and rev_{i+1} are identical, so the changes in rev_i must have been completely undone. These reverts are common in normal Wikipedia edit practice.[73]

This algorithm, however, is limited in its scope, and we may come across situations where reversions occur over a series of edits. Although the system may easily be extended to cover larger spans of history, to consider many nodes like this would require the edit-distance calculation of very many different node pairs ($0.5n(n-1)$ combinations for n entries; the edit distance relation between pairs is symmetric).

We propose a more efficient way of characterising redundant entries in terms of longer history spans. We must utilise the fact that we have take one article to be the ultimate destination of all previous edits in order to do so. Let us graph the entirety of a Wikipedia's revision history in terms of the edit distance from this final version (see figure 3.2).

Each point represents a different version, it's coordinates being its time-stamp, and the edit distance from that revision to the final revision. A line between two points represents the difference in to-final edit-distance between each version. Given this information, we may consider only those revisions that bring us closer to the final version. They appear in figure 3.3a as lines with a negative gradient (blue); those with a positive gradient take us further away from the final version (red).

We need not compute edit distances with a positive gradient (the two red lines). This is simple to implement. Given a version rev_i, having computed it's immediate edit distance $ed(rev_{i-1}, rev_i)$, and it's edit distance from the final version, ed_{final_i}, we know our next computation must be $ed(rev_{j-1}, rev_j)$, such that j is the smallest number that satisfies the qualities $j > i$ and $ed_{final_j} < ed_{final_{j-1}}$.

Another possible strategy would be to disregard all edit distances that, at any point, lie between two versions that are further away from final version than the version currently being considered.

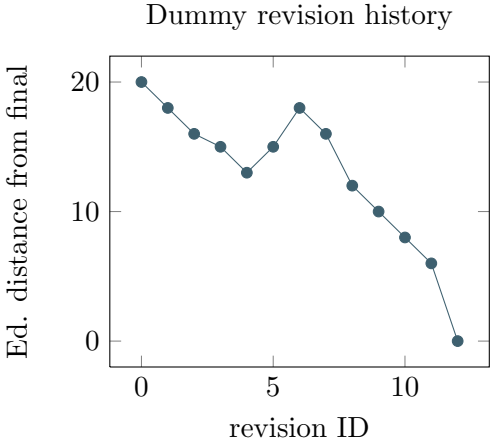
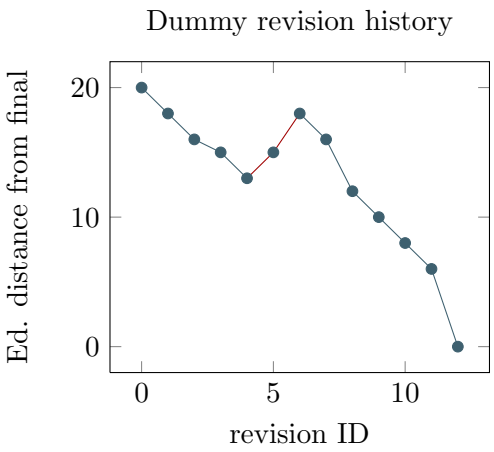


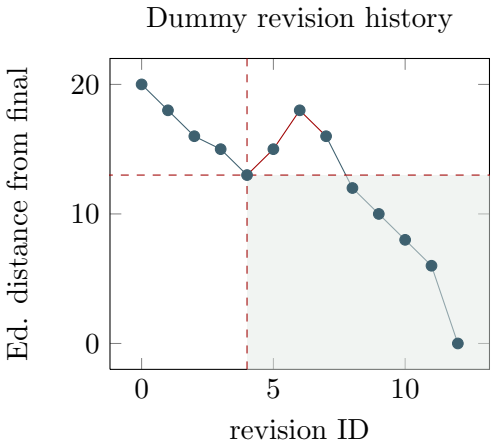
Figure 3.2: Graph showing a 'trajectory plot'

In figure 3.3b, rev_i, the currently considered version, is represented by the node at (4,13). The green rectangle shows the area in which we look to find rev_j, and the blue lines are those edit distances we both to compute. In this case, after considering rev_i, we move to the rev_j such that j is the smallest number that satisfies to qualities $j > i$ and $ed_{final_j} < ed_{final_i}$. In this graph above we move from (4,13) to (8,12).[2]

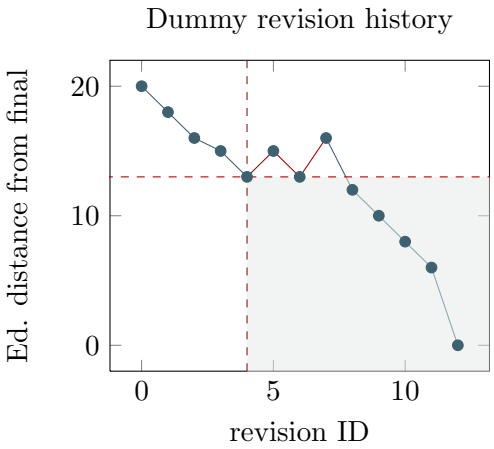
[2]It is worth stating here that, in both these strategies, after discovering rev_j, we always compute $ed(rev_{j-1}, rev_j)$ rather than $ed(rev_i, rev_j)$.



(a) Identifying discarded revisions



(b) A more 'lossy' approach



(c) A more complicated example

Figure 3.3: Trajectory plots with different features

However, with this technique we find some limitations – we may find a history such as the one in figure 3.3c. Between time-points 5 and 6, we find a downward trajectory, and we may infer that the editor has altered the article so as to more closely approximate the final version. Perhaps we should not simply disregard this edit simply because it is undone in the next revision.

Also, perhaps we may give some lessened reward to edits that move away from the final version, rather than disregarding them altogether. We may imagine quite easily that the work done here may have inspired 'correction', or at least be part of an ongoing discussion which moves development of the article on if not towards its final version.

So, rather than discarding the revision altogether, we calculate a 'gradient factor', which we may use to modify the reward given to an edit. Taking the trajectory distance as Δy, and the time difference between that revision and it's predecessor as Δx, we calculate a real number between 0 and 1 using the function in 3.5b. We map every point on a the shown arc to a our gradient factor.

The real number increases the more acutely the article approaches it's final version: inserting a lot of text that is eventually deleted results in a small gradient factor, and including a lot of text that is also in the final text results in a number closer to 1. By taking time frame into account, we define that the sooner an edit is made after the previous one, the closer the gradient factor will be towards either 0 or 1, depending on the sign of Δy. (Note that the gradient factor a negative Δx is undefined, as a revision cannot occur before it's predecessor.)

The significance of this number is two-fold – it at once describes the amount of change created in the history in terms of the target of that history, and the longevity of the article's previous state, (i.e. it's stability). For an edit to approach 1, it needs not only to contribute a large proportion of the text's final form, but also to do so very quickly after the previous revision. Later we will discuss what we can understand from these values in isolation, and how they relate to one another as a set.

We can also note that a repeated insertion may accrue an reward multiple times. We examine

dummy history found in figure 3.4.

We can imagine the simple case of two editors warring against one another – one inserts “fork” at time point 3, the other changes it to “spork” immediately afterwards. Then the original editor changes it back to “fork”, and so on. We see that “spork” was the favoured choice overall, as it receives a downward trajectory throughout. The “spork” user is awarded more that the “fork” user throughout, receiving a higher gradient factor for the downward-trajectory that the spork edits incurred in the trajectory. Both users are rewarded for having taken part in the discussing, though the ‘winner’ is rewarded more overall.

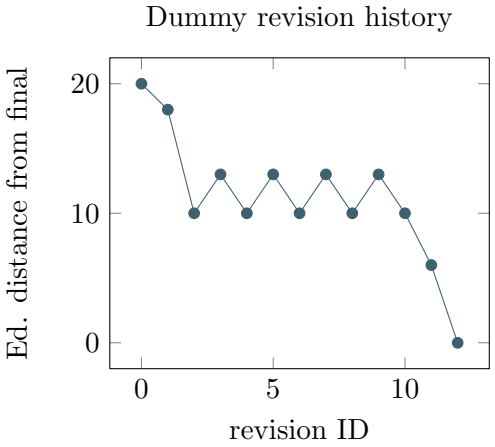


Figure 3.4: Graph showing trajectory with repeated actions

We can also imagine a very low value as a punishment for not allowing the previous version, which was closer to the final state, to exist for very long. The same edit made much later allows the preferable state a longer existence, and the gradient factor would be accordingly higher.

The trajectory-plotting technique gives us a simple and powerful way of visualising the history of these articles. Several good attempts have been made to visualise Wikipedia history data, with varying levels of success.[10][52][60][80][64] The most successful was the Wikipedia History Flow software,[41][42] which used line diff to keep a track of who inserted which lines. We believe the technique used here is more appropriate for the task at hand.

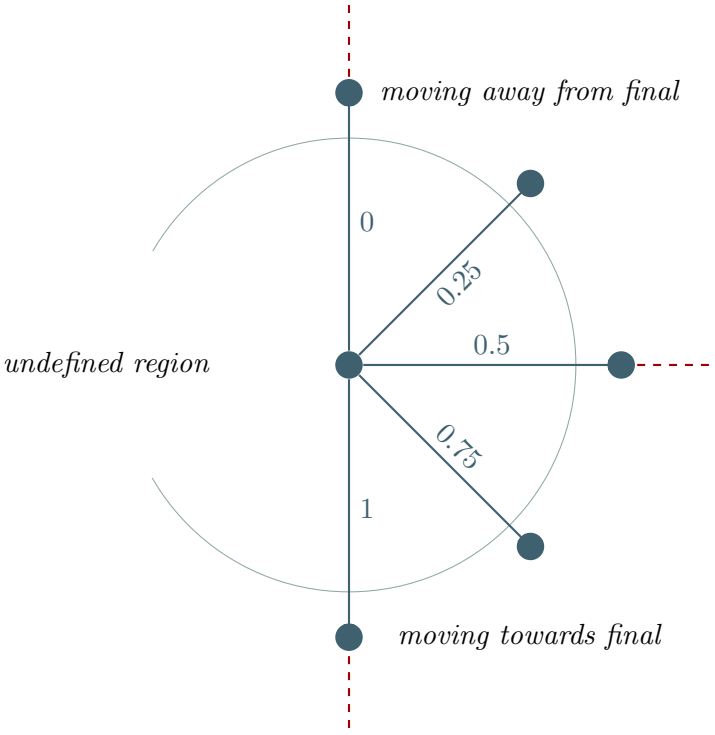
We can also use these techniques to widen our analytical scope. Past studies have used revision histories in order to figure out a variety of different things, including a 2012 study which used it to predict box-office success,[35] a 2009 study that was able to geo-locate editors by their edits.[26] We see that we can combine edit histories of different pages to visualise and analyse a more complete context for each article.

We will see later that we can use the same visualisations to identify some bot-work,[3] as well as other strange history features.

[3]It has been noted that there are around 700 bots registered on Wikipedia (as of 2014). Though not all of them make edits, those that do are very prolific, and are known to reverse malicious submissions in a matter of seconds.[75][37]

$$gfactor(\Delta x, \Delta y) = \begin{cases} 1 & \text{if } \Delta x = 0 \text{ and } \Delta y < 0 \\ 0 & \text{if } \Delta x = 0 \text{ and } \Delta y >= 0 \\ \frac{arctan(\Delta y / \Delta x)}{\pi} & \text{if } \Delta x > 0 \end{cases}$$

(a) Gradient factor definition



(b) Gradient factor definition: a visual representation

Figure 3.5: Mapping of trajectory angle from a single revision point to gradient factor: definition and visual representation

Identifying and splitting by text species

Using the wikimedia markup system, we may easily identify certain species of text. They are as follows:

- Internal links
- External links
- Images
- Files
- Musical scores
- Math-formatted text (similar to Latex math environment)
- Section headings (of differing levels)
- 'Citation Needed' tags
- 'As of' tags (used for identification of age-sensitive information)
- Block quotes
- Tables

We then group them logically, as follows:

Equation
: Math-formatted text

Source validation
: Block quotes, Citations, 'Citation Needed' tags, 'As of' tags

Links
: Internal Links, External Links

Structural
: Section headings, Tables
It was found in 2005 that this, if anything, was the clearest difference between Wikipedia and commercial encyclopedias,[18] supporting previous conjecture.[13] The regexes in this group provide a simple way of noticing changes that affect structure.

We may split the string along the boundaries of these markup tags, and send levenshtein distance calculator sperate strings, as shown in figure 3.6.

With these text species grouped, we m. By characterising an edit with a series of different edit difference we also perhaps create the opportunity to consider some as more valuable than others.

The algorithm that was settled upon left the levenshtein calculator itself naive of text species – instead we simply split the text up and calculate levenstein distance separately. We traverse each string from beginning to end, using simple regex expressions to identify and extract different kinds of text, and calculating the levenshtein distance for each separately. This process is detailed in algorithm 4.

In practice, this algorithm was much quicker than trying to add an awareness of text species to the levenshtein calculator itself. We using regex statements, we can search and split the string relatively quickly, and use this preprocessing to alleviate the levenshtein distance calculator of the burden of being aware of the kinds of text it is dealing with. With this awareness integrated, because

Algorithm 4 Revision pair distance calculation

```
regexes ←{
        'math1': '<math>((?!<\/math>).)*<\/math>\S',
        'math2': '{{math((?!}}).)*}}',
        'bquote': '<blockquote>((?!<\/blockquote>).)*<\/blockquote>\S'
        ...
}                                          ▷ Regexes that recognise single Wikimarkup tags
reggroups ←{
        'maths':(regexes['math1'], regexes['math2']),
        ...
}                                                  ▷ Group of regexes by 'species'
distances ← ∅
function PairDistance(str1, str2)
    strs ← [str1, str2]
    for all key, reg ∈ reggroups do
        comparestr ← ["", ""]
        for i ← 0, 1 do
            matches ← reg.matches(strs[i])
            for all m ∈ matches do
                match, strs[i] ←extractsplit(m.start, m.end, strs[i])
                comparestr[i] ← comparestr[i] + match
            end for
        end for
        if length(comparestr[0])> 0 OR length(comparestr[1])> 0 then
            distances[key] ← LevDist(comparestr[0], comparestr[1])      ▷ See algorithm 6
        else
            distances[key] ← 0
        end if
    end for
    distances['norm'] ← LevDist(strs[0], strs[1])                  ▷ Process the remainder
    return distances
end function
```

levenshtein distance considers one character at a time, these operations of flagging and identifying areas of text were inevitably multiplied many thousands of time in one operation. Instead we were able to use a fairly simply algorithm for calculating levenshtein distance, found in algorithm 6.

We will discuss different levenshtein-related algorithms further on this thesis, but for now we can say that the one we reference here is fairly basic, but with an optimised space efficiency. We see that we don't hold a whole matrix for the two strings, only the current and previous row. We may also describe the PairDistance overall as a divide-and-conquer algorithm. It improves the space complexity of the algorithm a little, and allows us the employ parallel or threaded processing in order to improve efficiency of computation.

Our procedure for collecting these values is described in algorithm 7.

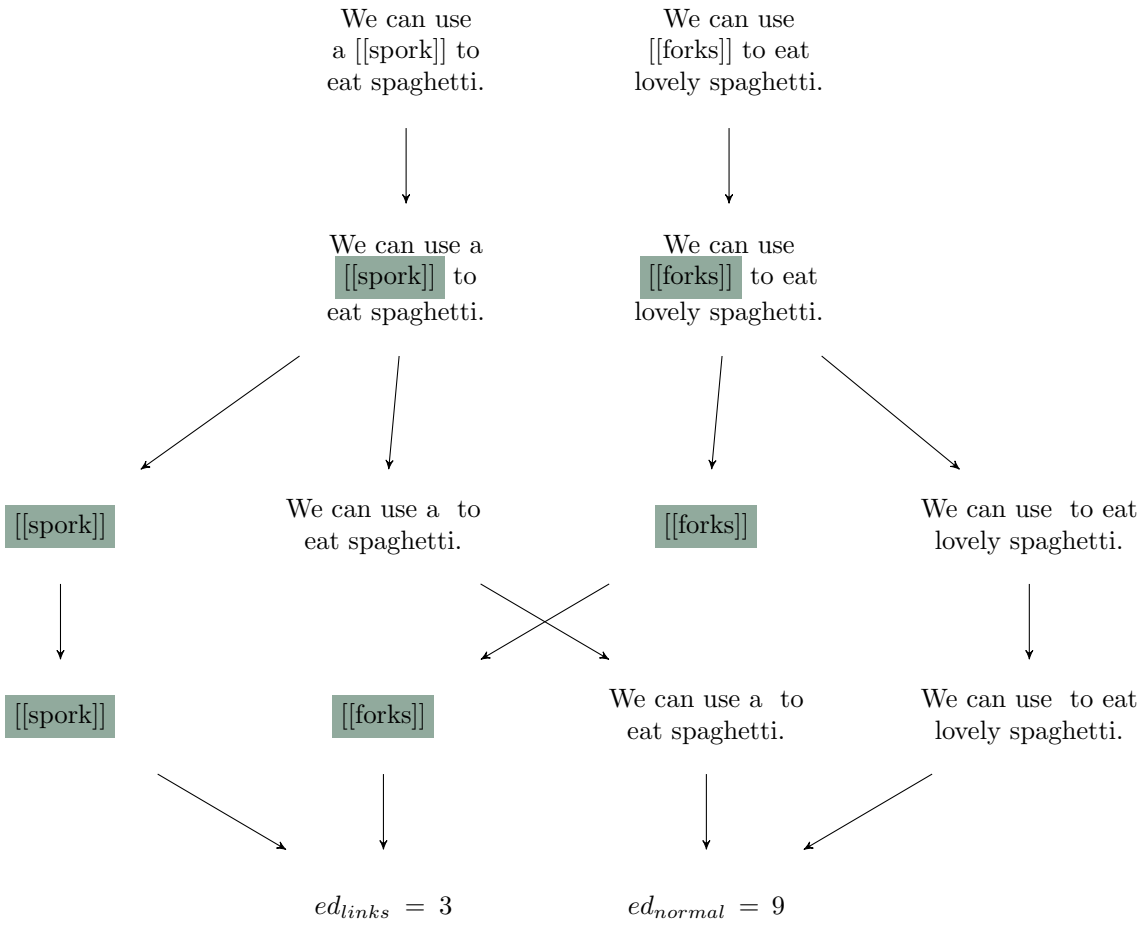


Figure 3.6: Diagram identification of link text, splitting text into normal and link segments, and performing separate levenshtein distance calculations

Algorithm 5 Pair comparison

```
    procedure PairComparison(revids)
        for i ← 0,length(revids) do
            if pair distance not already in database then
                str1 ← "" if i = 0 else database.gettext(revs[i − 1])
 5:             str2 ←database.gettext(revs[i])
                dist ←PairDistance(str1, str2)                ▷ See algorithm 4
                databaseinsert.pairdistanceinsert(dist)
            end if
        end for
10: end procedure
```

Algorithm 6 Levenshtein distance calculator

```
function LevDist(str1, str2)
    s1len ←length(str1)
    s2len ←length(str2)
    column ← [0_1, 0_2, ..., 0_s1len]
    for x ← 1, s2len do
        col[x] ← x
    end for
    for p ← 1, s1len do
        column[0] ← p
        r ← p − 1
        for q ← 1, s2len do
            oldnum ← column[q]
            column[q] ← min(col[q] + 1, col[q − 1] + 1, r + str1[p − 1] ≠ str2[q − 1])
        end for
    end for
    return col[s1len]
end function
```

Algorithm 7 Page trajectory calculation

```
procedure TrajectoryCalculation(revids, domain)
    target ←database.gettext(revids[−1])          ▷ Last revision in list is most recent
    for i ← length(revids), 0 do
        if trajectory distance not already in database then
            str1 ←database.gettext(revids[i])
            dist ←LevDist(str1, target)                          ▷ See algorithm 6
            database.inserttrajectoryinsert(dist)
        end if
    end for
    for i ← 0, length(revids) do
        dist2 ←database.gettrajectory(revids[i], domain)
        dist1 ← database.gettrajectory(revid[i − 1], domain) if i ≠ 0 else 2 × disty
        time2 ←database.gettimestamp(revid[i], domain)
        time1 ← database.gettimestamp(revid[i − 1], domain) if i ≠ 0 else timex
        Δx ← time2 − time1
        Δy ← dist2 − dist1
        if Δx > 0 then
            gradient = arctan(Δy/Δx) / π
        else if x = 0 then
            gradient = 1 if y < 0 else 0
        end if
        database.insertgradient(revid[i], domain, gradient)
    end for
end procedure
```

Chapter 4

Implementation

4.1 Technologies

Out first stop was to implement a library for tracing a Wikipedia history, harnessing the Wikipedia API in order to download information about articles and their histories, as well as their content. It is inspired by open Wikipedia metadata classes such as 'Wikipedia Miner'[15], or the revision-fetching 'Java Wikipedia Library / Wikipedia Revision Library'.[14][17] The python package 'wikipedia',[50] was also a useful starting point, but was not appropriate for the project in general.

The Levenshtein distance calculator, being an intensive computation, is implemented in C++ for speed purposes. In order to import to Python, we prepare python variables within the C++ code, and compile and build a shared object library as defined in the Python documentation.[49] To further improve speed, we deploy each levenshtein distance calculation in its own process, to allow for parallel processing on multi-core machines.

The database is implemented in PostgreSQL, with a python wrapper written using the psycopg2 module.[47] The text is automatically compressed on insertion – this is the default in PostreSQL.[45]

The graphs are outputted using the matlab-inspired python package, matplotlib.[30]

4.2 Guide to the code

The project was implemented in Python, with a small C++ core for the Levenshtein distance implementation. The different classes are fairly independent, but share the WikiData class, and are coordinated by the CLI classes.

To compute Levenshtein distance we employ a C++ script that returns Python-variables, and is compiled into a shared object library for use in Python. The Levenshtein distance calculations can be quite slow with long strings, particularly when calculating trajectory as the most recent revision can often be the largest, and we run the algorithm on that each time. Pair-distance can be a little quicker generally, and we are able to implement some multi-processing because of the string-splitting.

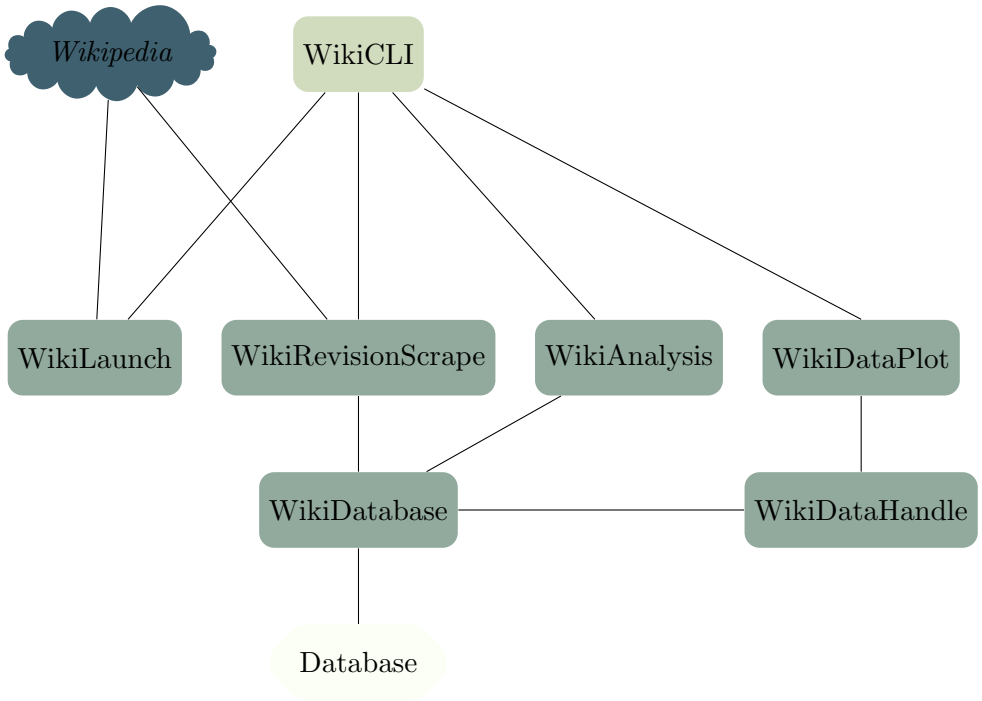


Figure 4.1: Diagram showing the connections between entities in python implementation

Otherwise, the code runs reasonably quickly, with it's speed most often limited by pthe HTTP requests, and the speed of the database.

This majority of this software requires PSQL and the psycopg2 package to run. The machine-learning script requires numpy and sci-kit learn.

Here we discuss the five main classes in the project, and their function in this project. Appendix A details the file structure of the code base.

WikiData

The WikiData class interfaces with the database, and maintains its integrity. It is importable as a package by the root-folder functions by the command 'import database'.

The functions here are as simple as possible, mainly basic fetches, inserts, with some alter functions. Each function has the same plan: check if necessary that the operation is needed, collect SQL string, collect data for string substitution, try to execute, if the execute worked, get the results, if the results are well-formed, return them in the correct format.

The functions towards the bottom of the file are dump functions, performing more complex operations for the plotting and machine learning scripts discussed below. They are harder to maintain, but are much more efficient than executing a combination of the other functions.

DHandler

This code fetches from the WikiData class, and arranges the results into plottable values for WikiDataPlot.

wikiDataPlot

The wikiDataPlot file can be run as a script, or instantiated as a class.

Running as a script accesses the dump-plot functions, fetching and plotting a lot of data from the database at once. It is controlled by a few command line arguments which are apparent in the code. This mass-plotting was really only implemented to help the writing of this report.

Otherwise, the lineGraph, barChart and trajectoryGraph functions are self-explanatory. These files will be output from the CLI given the *-p* option, and inserted either into the default file location, or a folder specified at the command line.

wikiFetch

This class implements algorithm 4, including all the necessary logical extensions to deal with CLI parameters. It interfaces with Wikipedia through various language sub-domains. It will attempt to fetch one article, trying until it succeeds if asked to choose a random article. Once it has found a file, details of the fetched page can be queried using function like getPageID, etc.

The class automatically disregards any article with less than 50 revisions, as we found analyses more worthwhile with longer histories. This can be overridden on instantiation of the class.

The language sub-domains are kept in a .csv file, scraped from the English Wikipedia, from which it picks and checks domain name from that if asked to.

The search function also attempts to suggest a title to the user if a page request fails, (though the suggestions are often pretty inappropriate...) using Wikipedia API functionality.

WikiAnalysis

An implementation of the algorithms 4-7. We employ multiprocessing to speed the computation up on multi-core systems, computing levenshtein each separate species of text in a separate process – each of the different edit distances calculated in the process in figure 3.6 is given it's own.

Conceptually, the child processes are spawned at line 16 of algorithm 4, and the results collected just before line 22. As shown in figure 4.2 the 'normal' text portion comparison is inevitably longer than any non-normal, and we execute this thread last (it being the remainder text), so we are mostly waiting on this process for a while before the computation can proceed. Nevertheless, we found a notable increase in performance by implementing multiprocessing.

The Database

The database is implemented in PSQL, and accessed via our python package, 'wikiDatabase'. The package leverages the psycopg library, which allows simple access to the SQL database via python.[47] The package is used by all the other classes used in this project and provides simple inserting, changing, fetching and checking of data.

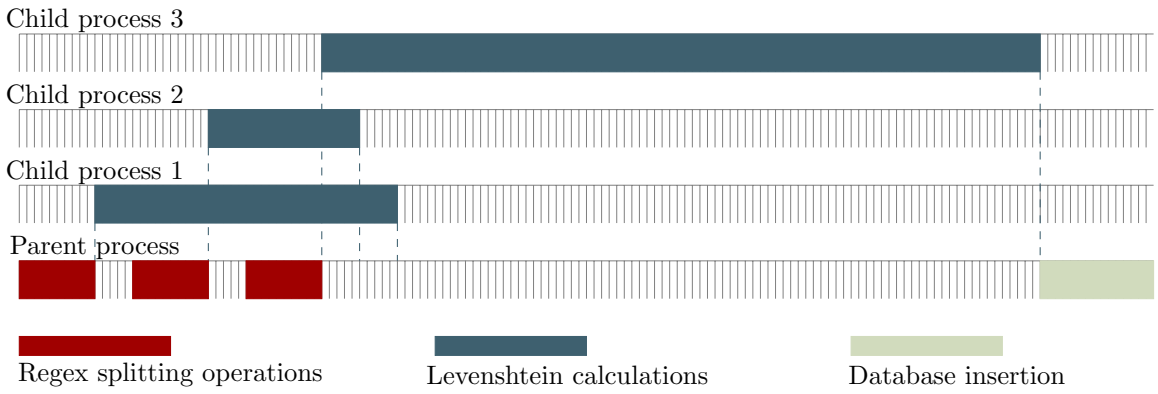


Figure 4.2: Diagram showing multi-processing in pair distance calculation

revid	domain	content
⋮	⋮	⋮

(a) Table: wikicontent

pageid	domain
⋮	⋮

(b) Table: wikifetched

revid1	revid2	domain	distance
⋮	⋮	⋮	⋮

(c) Table: wikitrajectory

revid	domain	pageid	title	username	userid	time	size	comment
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

(d) Table: wikirevisions

revid	domain	maths	citations	filesimages	links	structure	normal	gradient
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

(e) Table: wikiweights

Figure 4.3: Schemata for the database used to store wikipedia data

Some more complex operations are undertaken by the class, mainly preparing large 'data-dumps' for mass-plotting (in preparation for this report) and validation exercises. These functions were written for performance purpose (the data they deliver could easily have been fetched using the other functions, but would have taken much longer), and are clearly marked in the code as separate from the basic fetch, get and insert functions.

The database relies upon the uniqueness of revision ID and domain-name, and page ID and domain-name pairs. pairs to function. The existing database schemata can be found in figure 4.3. We talk about improving this schemata in the evaluation chapter.

A database backup can be found in dbdump.sql, and a script for setting up the schema in schema.sql.

wikiLaunch

A simple collection of functions for launching Wikipedia in the browser, able to target diffs between two revisions, old revisions, pageids, and user pages and all Wikipedia sub-domains.

The CLI interface

The CLI can be accessed using the short-cut wrhp (Wiki Revision History Portal), and offers a limited, automated coordinating and run of the above classes. It can be manipulated using the arguments as follows:

Default behaviour. Fetches a random page from a random Wikipedia to the database, and analyses it.
-s Scrape only.
-p Plot data. Saves png files to location given by the –plotpath argument.
-i Open the interactive plot window for a given (or random) article once analysed.
-v View a Wikipedia page online. Must be used with –domain. Can be used to view a diff (using –revid and –oldrevid), a specific revision (using only –revid), or the latest version of a page (using –pageid).
-t 'Trundle' mode. Repeats the given operation until interrupted. Useful for building up a database. Cannot be used with the –titles argument.
-S[S] 'Silent' mode. One 'S' silences stdout, two silences both stdout and stderr.
–**title** [***str***] Specify the pages to be scraped. Must be used with –domain. Case (and spelling...) sensitive.
–**domain** [***str***] Specify the domain to connect to. May be used without –titles, limiting the random page pick to one domain. Must be the short version ('en', 'de', etc.).
–**scrapemin** [***int***] Specify the minimum amount of pages to be scraped for one page. Default = 50.
–**plotpath** [***filepath***] Specify location for plots to be stored. Default = .
–**revid** Specify the revision ID to be viewed. Used with -v.
–**oldrevid** Specify the old revision ID when viewing a diff. Used with -v.
–**pageid** Specify a target pageid. Can be used in -v and -s.

Chapter 5

Results

The basic output of the model described here is a set of numbers, a levenshtein distance for each of the groups described in line 2 of algorithm 4 (and described more clearly by the schema in figure 4.3), and a trajectory factor, calculated by the procedure found in algorithm 7. Accompanying this is the trajectory distance of each revision.

Through simple database queries we may also find out other related information, such as how many edits an article has, how frequent these edits were, and the article's final or average size.

With a large enough random sample, we begin to be able to report the relative activity of individual editors also. If, for every page 'X' we also look for and grab the page 'Talk:X' we may see which editors were involved in discussions around the edits, and begin to more clearly see the context surrounding each article. We can also look for pages named 'Wikipedia:Requests_for_arbitration/X' to discover evidence of more serious disputes. The CLI does not implement this automatically, but we will discuss how to combine this information automatically and what we find out from it.

Using the DataPlot class is built to output four different kinds of graphs – trajectory graphs (trajectory against article growth), a trajectory vs talk page graph, a bar chart grading users by edit count, and a similar graph grading them by the 'share' of the article. This latter graph can be modified by the application of weights. The weights allow us to take some species of text as more important than others when calculating a user's share in the article.

The DataPlot class can also output graphs on the database as a whole. Running it as a script we can plot information about how the size of articles in each language, the lengths of their histories, etc.

Analysis

Share and edit count plots

We find two typical, unweighted edit graphs in figure 5.1.

These two graphs are typical of the data set on the whole, showing many one-time editors, with a small amount of repeat editors. More similar graphs are found in appendix B.

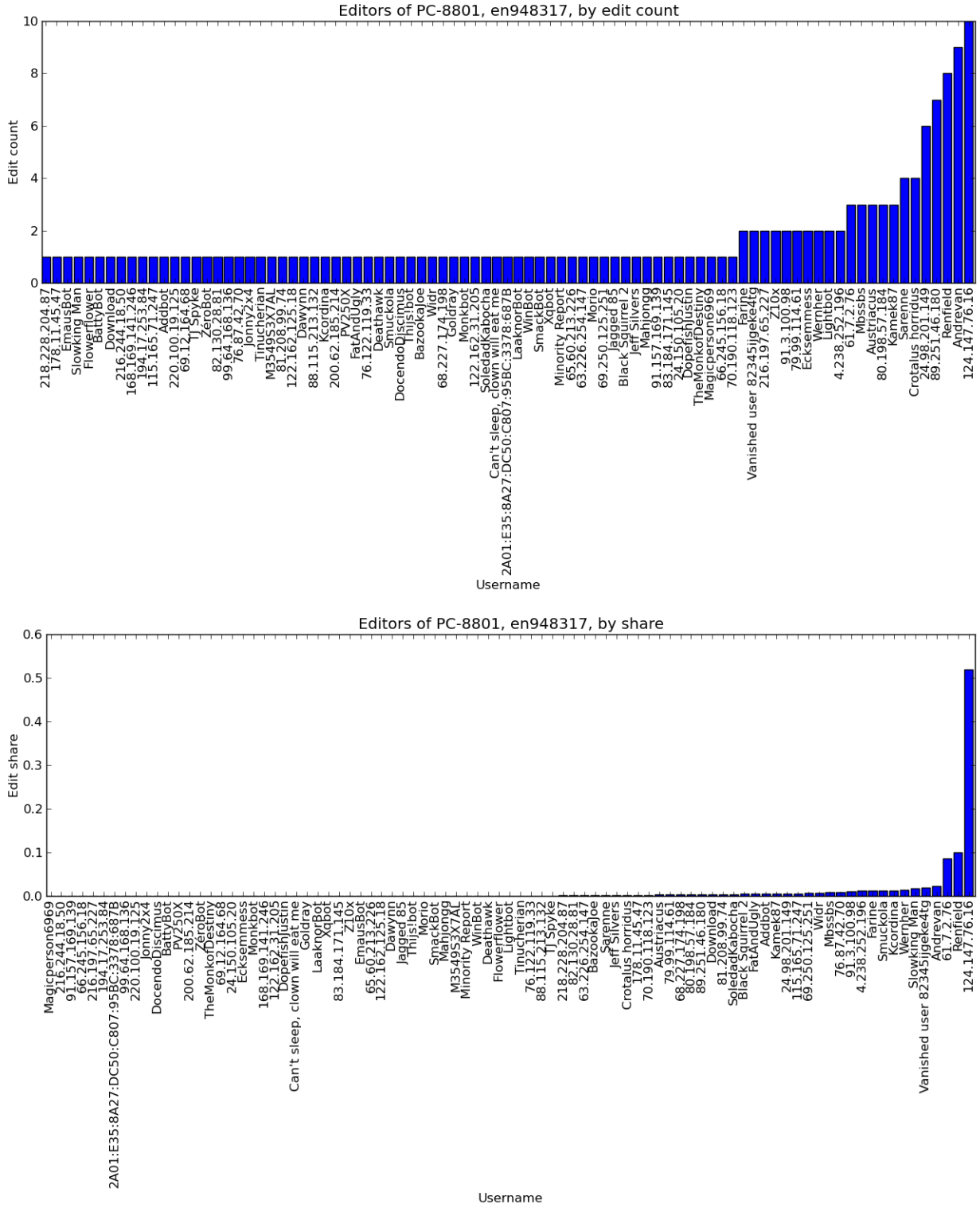
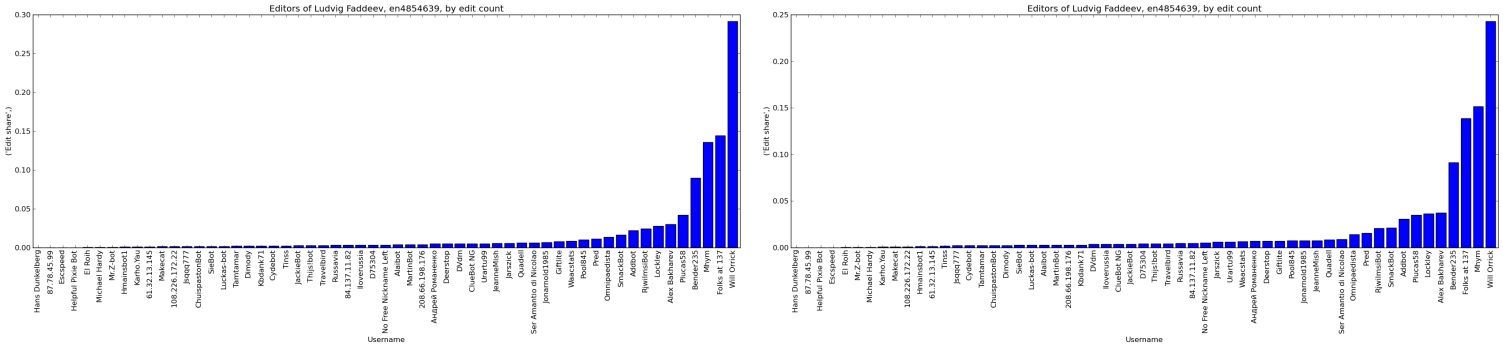


Figure 5.1: English Wikipedia, Page ID 948317 (PC-8801), user graphs



(a) Unweighted share

(b) Share with doubly-weight links

Figure 5.3: English Wikipedia, Page ID 4854639 (Ludvig Faddeev)

We can note here that a lot of the single-time entries are anonymous, showing an IP address rather than a user-name. Since IP addresses are likely to change (it is standard practice for a ISP to provide a new IP address for their customers on each new connection), we cannot know how many separate agents these IP addresses represent. We can't necessarily be sure that the 10-edit user is one person either, but for the purposes of this study this isn't an issue.

We can also find our first bots on these graphs, with many making small numbers of edits. EmausBot towards the left of the edit count graph, BattyBot next – it is common practice to name the bot account with an obvious bot-name. Emausbot comes up a lot in our analyses, in fact. It maintains inter-wiki links.

From our first graphs we found that many articles, particularly in smaller Wikipedias have little-to-no edits, and would result in graphs like the one in figure 5.2. For this reason, when fetching random articles, we automatically discard a revision if it falls below a certain threshold revision-count.

We also experimented with automatic, arbitrary weight setting to see how the graph would be effected. We see one such example in figure 5.3, where we contrast an unweighted data set, with one where we double the weight of the 'links' weight.

In this case, in fact, it was doubling the links value that created the most dramatic change. Changing maths, files/images, structure and normal text weights created the least change – all the graphs can be found in appendix D for comparison.

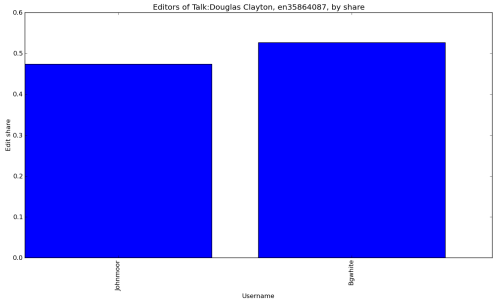
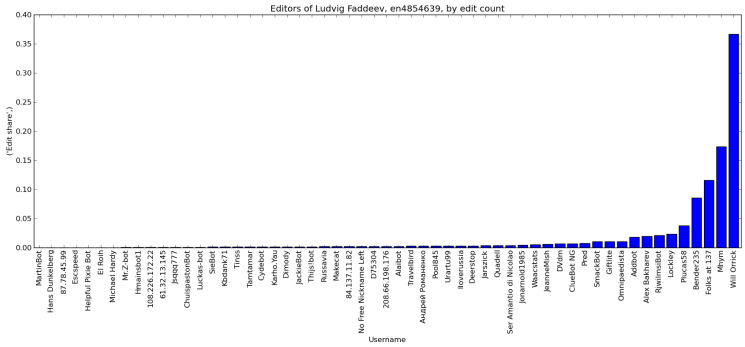
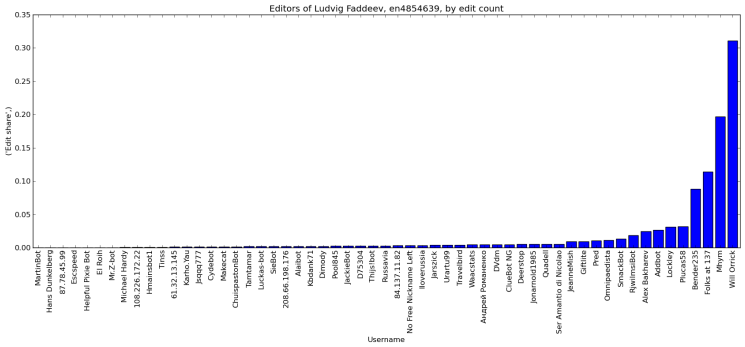


Figure 5.2: English Wikipedia, Page ID 35864087 (Talk:Douglas Clayton), share

All the graphs we have looked at so far do not take our gradient factor into account. The effect is interesting when we activate the gradient factor. Multiplying each summed weights by the gradient factor as we calculate share, the two graphs in figure 5.3 are transformed into those in figure 5.4. The graphs are notably skewed towards those with the most reward already.



(a) Unweighted share, with gradient factor



(b) Share with doubly-weight links and gradient factor

Figure 5.4: English Wikipedia, Page ID 4854639 (Ludvig Faddeev)

Trajectory plots

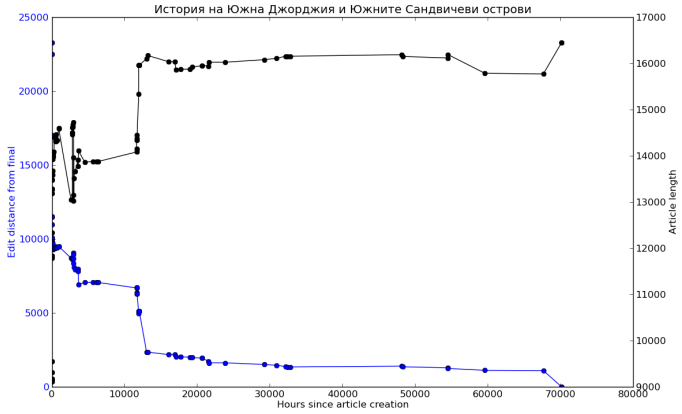
On average, these plots follow a similar pattern: the Levenshtein distance of the article begins at a high value at $x = 0$, it's creation date, and approaches the final article state at $y = 0$ and a high value of x. The article begins in a near-empty state, and as it grows, becomes more and more similar to the final version, before finally reaching 0 distance-from-final.

In the simplest cases, the growth of the article is a rough mirror-image of the distance-to-final, beginning at the origin and rising. A few of these 'classic' examples can be found in figure 5.5. These graphs show a reasonably steady approach of the final state, increasing in size in a fairly simple way.

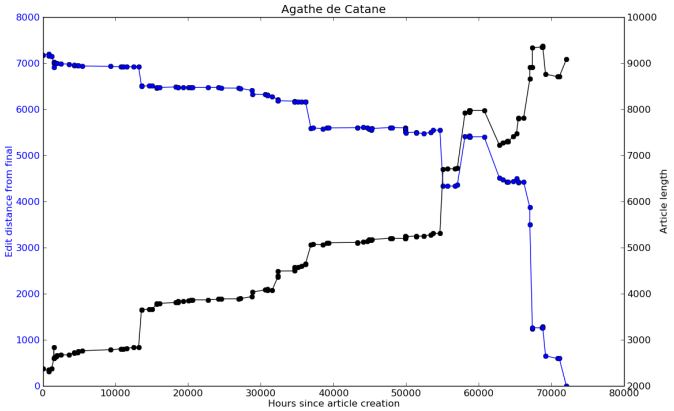
In the last of these simple trajectory graphs we find the first of our notable features. Figure 5.5f shows a clear sign of an undo operation at around 65,000 into the age of the article. We see the black line take a steep ascent and immediate descent, showing a quick enlargement and reduction of the article size. Since the same spike can be seen in the blue line, we know that inserting this text made the the article more different from the final version, and the removed text brought us back. We can infer that the same text that was added was the same as that which was removed. We find one of these redo-undo edits in the wikipedia diff between revids 96659084 and 96654283.

In contrast, the Phillipino article on the Nicene Creed (Figure 5.5d) has a similar-shaped spike in it's size. The event can be seen to occur just after 40000 hours. In this case however, the trajectory line does not spike in turn. In this case we can assume that the inserted spike shows an insertion of text that remains in the article, and the deletion of text that is not re-inserted before the article's current version.

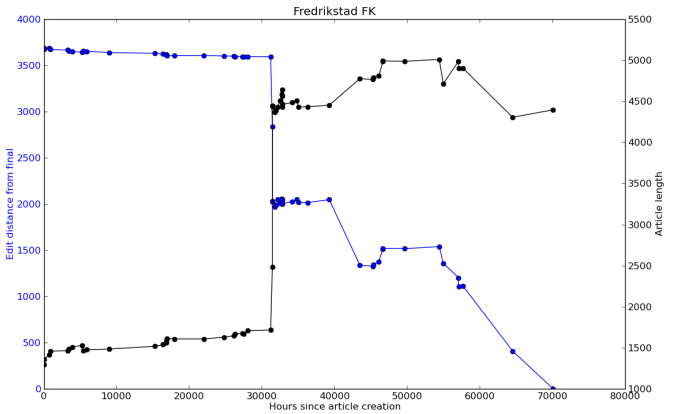
These spikes are fairly obvious characteristics to identify on plots such as these. However, we may also identify longer periods of meandering away from and back towards the current version. Notice the arced paths in figure 5.5c, spanning around 45000 to 55000 hours. Although the time frame is much larger than in the other articles (10,000 hours $\approx$ 1 year 2 months), and multiple edits occur in the meantime, we may also characterise the overall event as an 'undo' operation. It is more nuanced than others we have identified so far (after the event the article is equally similar to the current version, but smaller in size), but the effect is the same. The burden is upon our model as to how to sufficiently characterise these long-term removals of useless inserted material.



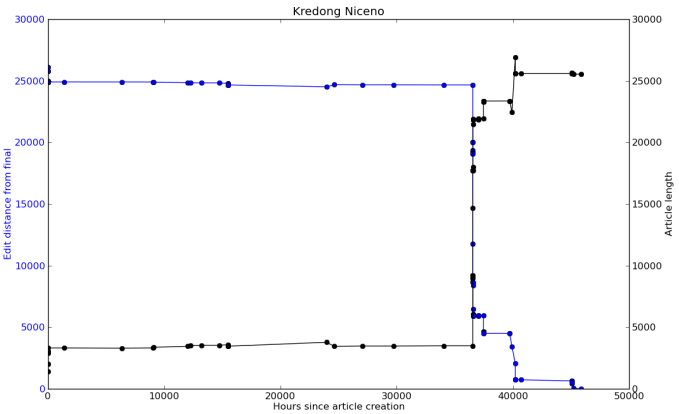
(a) Bulgarian Wikipedia, Page ID 90882 (History of South Georgia and the South Sandwich Islands)



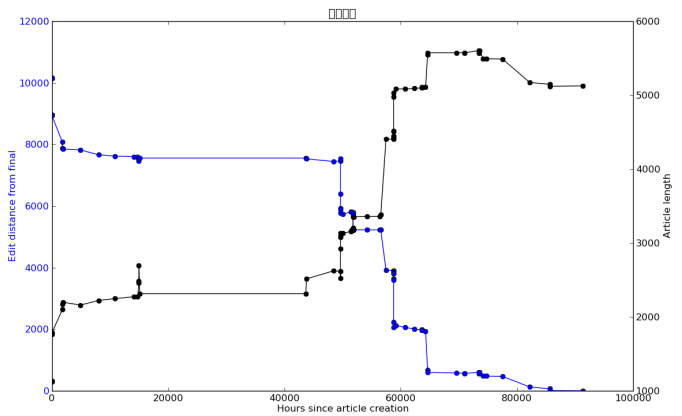
(b) French Wikipedia, Page ID 572796 (Agatha of Catania)



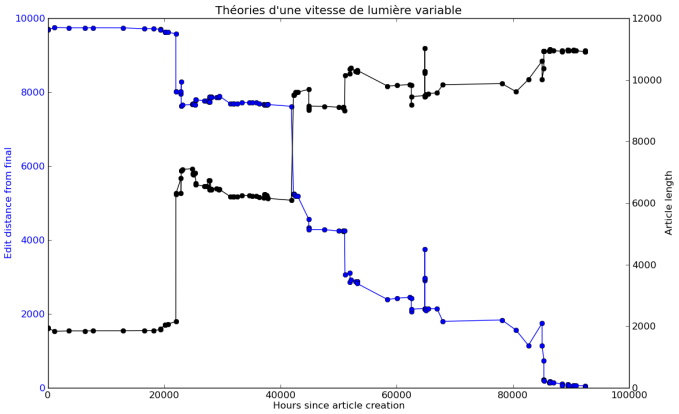
(c) Polish Wikipedia, Page ID 291709 (Fredrikstad FK, a Norwegian football club)



(d) Tagalog Wikipedia, Page ID 62286 (The Nicene Creed)



(e) Japanese Wikipedia, Page ID 1611162 (The Tax Treaty)



(f) French Wikipedia, Page ID 43937 (Theories of a variable light speed)

Figure 5.5: Simple trajectory graphs.

We also may begin to notice that that edits may tend to cluster in time, and that the articles may owe much of their content one-time bursts of activity. The Japanese article on tax (figure 5.5e), after an 'undo'-shaped cluster of edits at around 15000 hours, was not edited again for almost 3 years. The Tagalog and Polish articles (respectively the smallest and largest articles discussed so far) gained most their content over just a few days.

Stranger history can be found in the plots in figure 5.6. These show another common shape: a slow climb in both distance-from-final and size, before a sharp drop. The shape seems to describe an article which engorged with content, before having the majority of it deleted. On further inspection we found that the articles generally share two characteristics – they come from small Wikipedias, and the majority of the edits are made by bots. We can see from the user-names on the right-hand side that they are bots – there is a special bot 'flag' to be raised in the user profile of bots, but it is also traditional for the bot to be also be given an identifying name.[74]

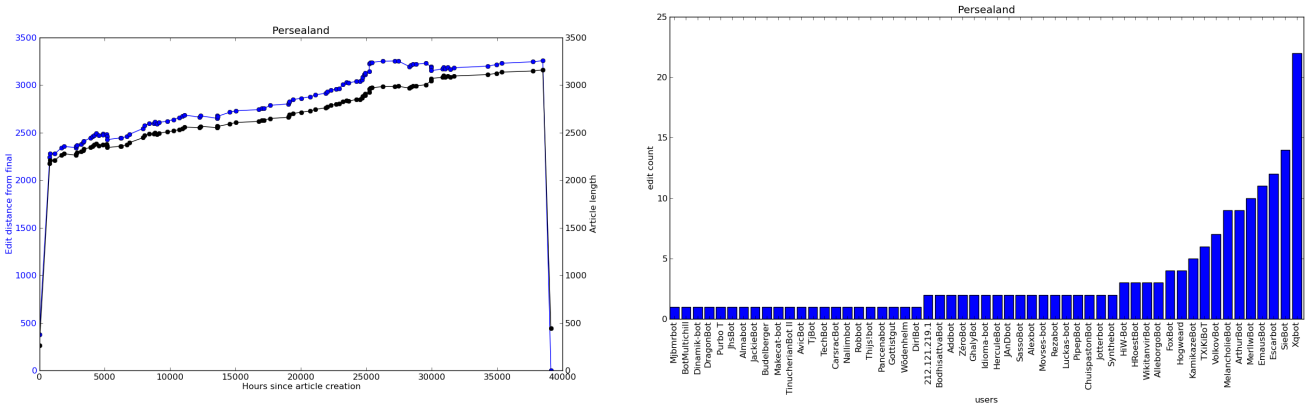
The explanation is in the articles' histories. In each one, we find a single large deletion event, occurring any time after early 2013. With a little digging, we find the origin of the act to be a Wikimedia project that aims to 'migrating inter-language wiki links from individual articles into a central database to ease maintenance',[78] see figure 5.7 for the native wiki diff of just one of these events. Each wikipedia article is rendered to the right of a language bar that links that article to its counterpart in various languages. Once coded into each article manually, these links are now stored in and served dynamically from a central source.[43] The move gives us strange results for this project – for small articles, the loss of these hard-coded links constitutes a gutting of its majority content. By our measurements, in these circumstances, this act of housekeeping is a momentous event in the history of small articles.

How do we characterise this? In this case the 'article', as such, is unchanged. Indeed the rendered page is identical before and after the link cull. We may think to ignore bot edits to a page, but how do we distinguish between the addition and deletion of these language links from meaningful changes to content? We may distinguish from, say, useful spell-checking bots by means of a simple text regex (inter-wiki links always have the same form), but we may not as easily distinguish language-bar links from in-line ones – those that may link to related articles, rather than mirror articles, widening knowledge rather than offering what may merely be a translation of the original article. We discuss this later.

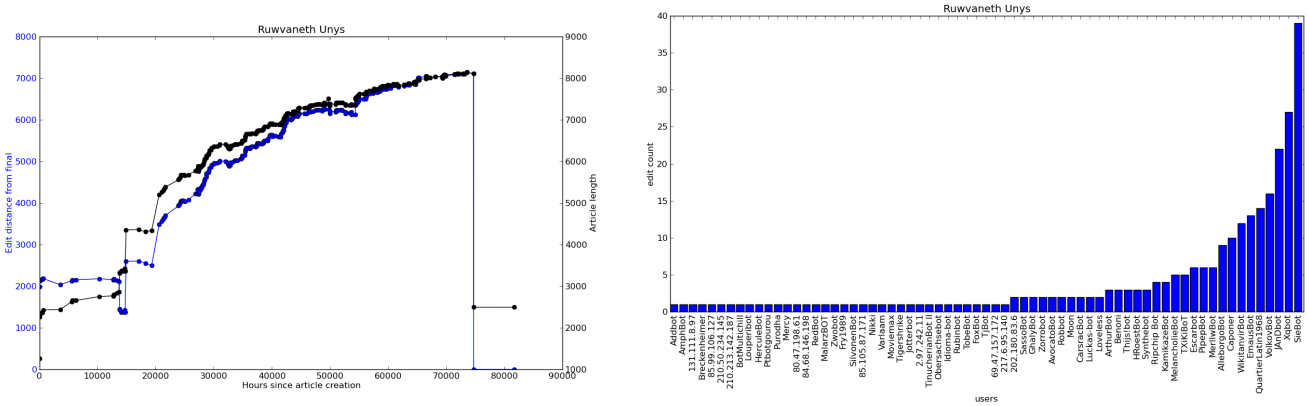
Combining trajectory plots

Derek Smart's wikipedia page has been named as the battleground of one of the 'lamest WikiWars' – it is the place where a very long argument about whether or not to include criticisms of his work on his page. We use it here as a case study for how derive more information about the context of a page with our tools.

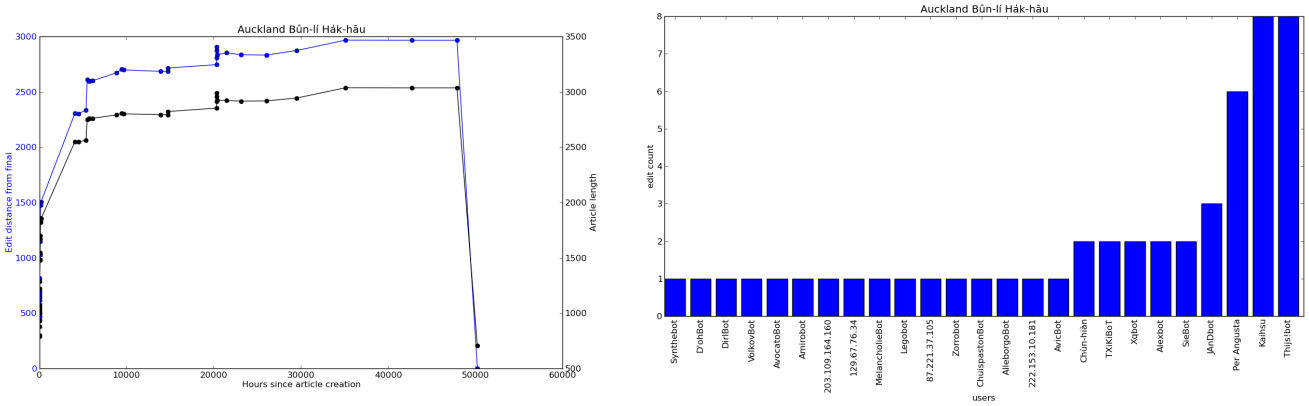
In figure 5.8 we see the combination of trajectory plots on three different pages – the green and black lines represent the same article trajectory / article size we've looked at previously, but accompanying those we have plots of the talkpage for that article, and the page detailing the arbitration request raised for that page.



(a) Anglo-Saxon Wikipedia, Page ID 7120 (Persia)

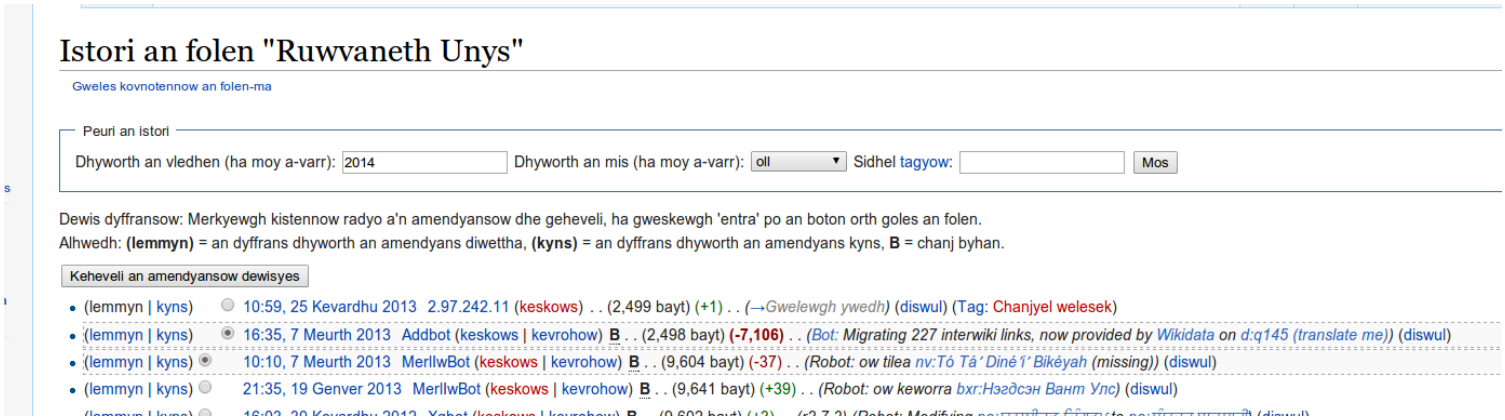


(b) Cornish Wikipedia, Page ID 726 (United Kingdom)



(c) Min Nan Wikipedia, Page ID 10733 (Auckland Grammar School)

Figure 5.6: Trajectory graphs showing bot editing characteristics.



(a) Migration event in article history (16:35, 7 March 2013)

(b) The Wikipedia diff for the migration event

Figure 5.7: Screenshots showing automated migration of inter-language links.

We see clearly page's activity density is relative to the density on the respective talk page. We also see that a large part of the article's history follows the same large-unnecessary-growth pattern we recognised in the small, bot-dominated pages analysed previously. By the end of the densest period, the page is much smaller, and is quite dissimilar from the article state at the height of activity.

But perhaps the most interesting part of this graph is the plot of the request for arbitration page – the blue line. We see it created at the height of the page's size and from-final dissimilarity, and during a particularly turbulent patch of the talk page history. We also see that as the arbitration page reaches it's final versions (the decision on the dispute), the commotion begins to cease, the page shortens, is edited less frequently, and approaches it's final version quite quickly.

In this particular case, a small handful of members were arguing about whether or not to include criticisms on Derek Smart and his work in his article. At the height of the argument, they involved parties called for an arbitration committee to help resolve the issue. The article was "urgently referred to the Wikipedia editing community at large for cleanup, evaluation of sources, and adherence to [Neutral Point of View]", as well as restrictions on who could edit the article, and how.

So perhaps it would be important for us to reward editors who participate in these discussions accordingly. With simple arithmetic operations we can see when an argument has turned out to be eventually irrelevant to a document's final version – we see that this one involved the addition of a lot of eventually removed text – and perhaps penalise the key editors involved by cross-referencing their edits between the pages. We can also see when the opposite occurs. The downward slope of the graph in figure 5.8 perhaps shows a reconciliation, and efforts to change the article according to the conclusions of the article. We may see if the arguing editors are involved in this 'tidying', and regard them accordingly.

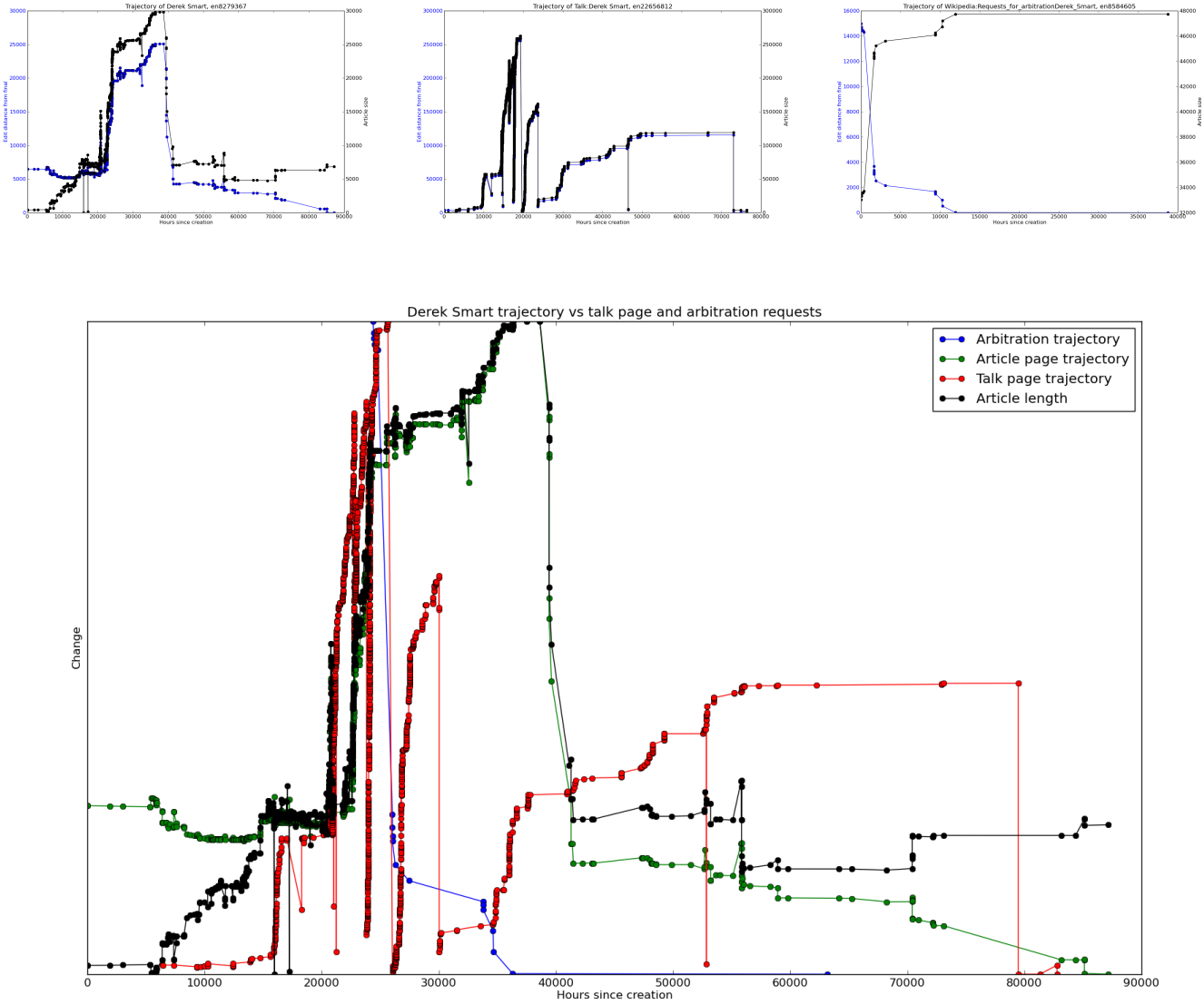
In any case we see that the immediate context of a page can have great effects on the path nature of the collaboration, as well as the final product.

We find a similar case in the biographical article for English biologist Rupert Sheldrake. Rupert Sheldrake, whose 'astonishingly visionary' early work in plant science had him described as 'one of the brightest Darwinians of his generation',[1][3], came under heavy criticism for his later work in psychical research, and 'mysterious telepathy-type interconnections between organisms and collective memories within species'.[54]

Figure 5.9 shows a graph similar to the Derek Smart plot, though a request for arbitration has not yet been made. In fact, though it's history is already quite dense, it has only been in the last year-or-so that the activity has really started, and we can expect this graph to look quite different if recalculated in a few months.

This spike in activity is a result of direct publicity. Rupert Sheldrake has been controversial for years (in 1981, Nature wrote of one of his works that it was 'a book for burning'), so it is not surprising to see arguments surface here.

But on Wikipedia, we may see that the activity peak is associated with something slightly different



(a) Cornish Wikipedia, Page ID 726 (United Kingdom), normalised y-axis values

Figure 5.8: Obtaining greater context of an article's history by combining page trajectories

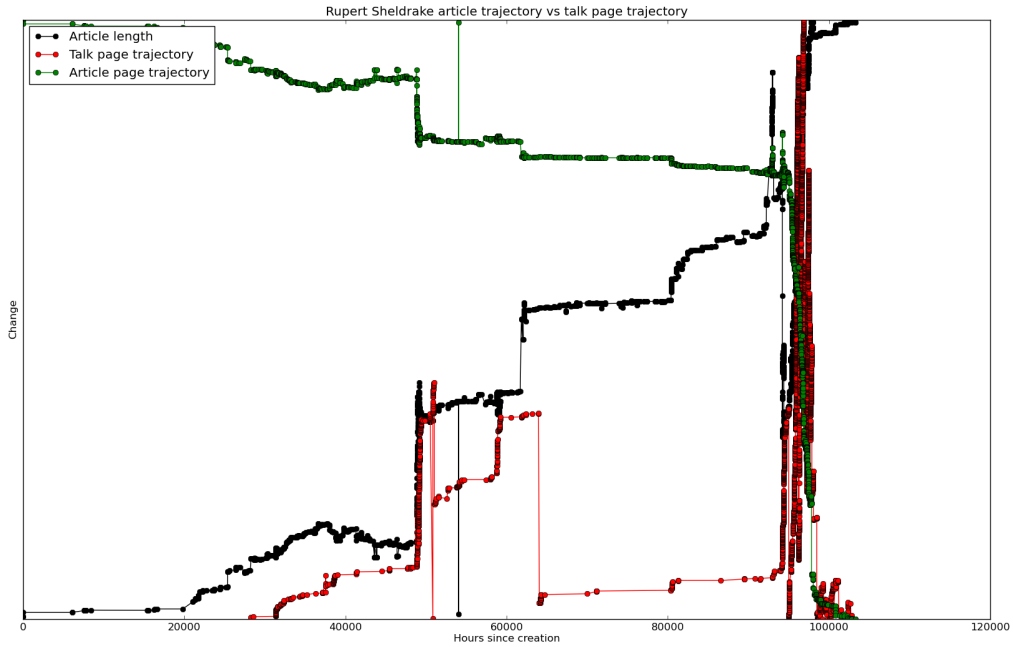


Figure 5.9: English Wikipedia, page ID 142395 (Rupert Sheldrake), article vs talk page trajectory graph

— provoking regular editors by questioning the their possibly overpowering predilections towards certain opinions. In November 2013, Rupert Sheldrake accused 'guerilla' editors of "distorting hundreds of pages on Wikipedia"[55], and the argument started from there. One of the key players in the ensuing argument on Wikipedia began his own website about the possible problem, in reaction to his percieved abuse. It is called 'Wikipedia, we have a problem'.[65]

We consider, then, that the context of these edits may not simply be edit count, density of talk, but a page may also be greatly affected the unpredictable whims of editors, that the trajectory of a page may be shaped by a conflict of opinions, rather than facts.

Chapter 6

Evaluation

6.1 Limitations, optimisations and extensions

Alignment of levenshtein distance

As we began to realise that, in terms of survival rate, the details of the content were somewhat secondary to the context in which those edits were made, we began to wonder if we could make the data analysis more efficient by doing away with the pair distance calculation.

Further than that, however, we wondered whether we could find the distance between each revision by just taking the absolute distance between each trajectory distance – i.e. the distance-from-final at each point. We saw no theoretic problem with the measure, and set out to see if the theory that

$$|(ed(rev_i, rev_x) - ed(rev_j, rev_x))| = ed(rev_j, rev_i)$$

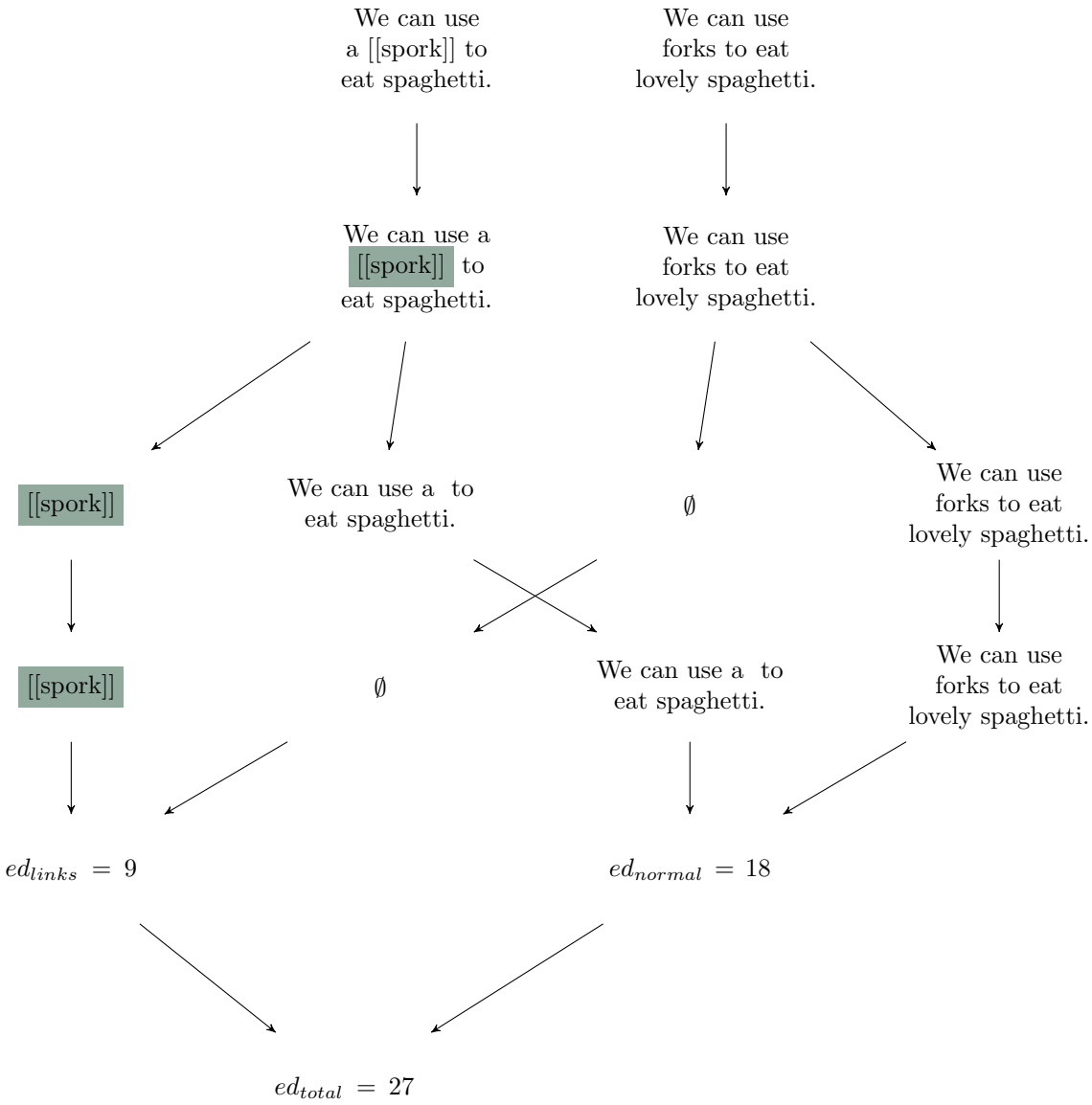
(i.e. that an edit distance relation is transitive) could be observed in the edits.

Running the SQL statements found in appendix E, we found evidence contrary to our theory. We found that only around 30% of our different measures matched up, with a high average mis-match of 22,181. The mismatch is, on average, around 40% of the larger distance.

We propose two theories for this. We can identify a problem with our string splitting – it may mis-align two strings, creating non-minimum calculations, such as in the diagram on page 7. We illustrate the problem in figure 6.1.

We can also imagine that it may be more efficient to swap some characters between a and c, rather than delete to them to get from a to b, and reinsert from b to c. This may also be described as an alignment problem.

Whilst inaccurate, doing away with pair-distance calculations may be acceptable as a rough measure, and if pair-distance, as we suspect, is not a very effective measure of survival.



(a) Split string edit distance

$ed($"We can use a [[spork]] to eat spaghetti.","We can use forks to eat lovely spaghetti.") $= 6$

(b) Whole-string edit distance

Figure 6.1: Showing alignment alignment inefficiency in split-string Levenshtein distance processing.

Calculating 'turbulence'

The gradient factor that we calculate describes how much a given revision changes in terms of the final state – it is a real number from 0 to 1, 0 being a 'perfect' move away from the final version, 1 being a 'perfect' move towards. On the trajectory graphs discussed these two values are represented by a vertical line between two points, going upwards and downwards, respectively.

We consider that we could perhaps use it to characterise a 'perfect edit factor' – one that approaches the final edit most efficiently; in a linear fashion. An optimum revision history may be considered as a straight path from origin to destination. No edit inserted text that was later removed, and the approach to the final version was as efficient as possible over time. We may characterise such an optimum gradient as follows:

$$gfactor_{optimum} = gfactor(ed(rev_{0_{content}}, rev_{n_{content}}), rev_{n_{tstamp}} - rev_{0_{tstamp}})$$

This equation gives a gradient that, if the same for every edit, would describe the most stable accumulation of data, with no peaks or fluctuations in activity. The average deviation from this number can measure the turbulence of an article's path.

We may also look to density of edit. If we have a set of the indexes of an edit operation as $\{op_0, op_1, op_2, \ldots, op_n\}$, where op_i is the index of the ith operation, then we may evaluate it's density with a standard deviation of the edit itself, σ_{ed}, multiplied by the span of the edit itself in context of the wider article. Something along the lines of:

$$ed_{density} = \frac{(op_n - op_0)\sigma_{ed}}{|v_{ed}|}$$

where $|v_{ed}|$ is the overall length of resultant version. By implementing this carefully, we may achieve a gradient of weighting, with a lower weight values for things like spell-checks, and higher values for whole-paragraph changes.

However, this density is somewhat reflected in the gradient factor measure, as discussed before.

Levenshtein algorithm changes

The Damerau-Levenshtein distance defines an 'swap' operation, which is the reversal of two adjacent characters. It is particularly suited to spell-checking, and for analysing DNA-sequence variations. In this case:

$ed_{damerau}$("ab", "ba") = 1

Implementing this could more accurately define spell-check operations.

Block distance operation would allow us to recognise displacements of entire blocks to count as one operation. For example:

ed_{block}("abcde", "cdeax") = 2

One move of the block ‘cde’, one substitution of ‘b’ for ‘x’.[62] We could implement this in order to better diagnose large movements of text — perhaps between documents. (The Harry Potter history in appendix C shows frequent migrations of text to book-, character- and film-specific pages.)

Awarding restructuring

It has been found that, even in the most accurate articles, that the structure of Wikipedia article can be weak.[18] We made some efforts to recognise attempts at structural change previously with our regex splitting system. However, these attempts are limited. We really need to be able to identify the move of large blocks of text – not only inside an article, but to others. With a larger dataset and the latter idea necessary computing power this would just be a matter of cross-referencing the deletion of largish chunks of text with insertions of similar text into other articleswithin a short time frame.

Refinement of gradient factor

A limitation of gradient factor is it’s reliance upon a relatively arbitrary measure of time. Though the measure of time has been consistent throughout the study, the measurement does affect the distribution of possible gradient factor values to the extremities of the range. It would be recommended in further research to take this into account. If someone makes an edit, commits it, fixes spelling, and commits again, does he receive less reward than if he had submitted at once? Or more? This is something we wished we would have had time to look into during the study.

Extending data

Grabbing each page as we went was useful for cleaning and sometimes preprocessing parts of the data, but the fetches were time consuming. Also, when we came to look for more general data about individual users, and tried to find inter-article connections, we had to do so in the knowledge that our databases represented a very small selection from the Wikipedia’s in question.

For further work on could be better off using a wikipedia dump. These are made monthly and represent the entirety of the site, history included at the point of export. They are, however, 800GB, compressed.[68] A past studies used a hadoop framework to enable the study of this data. In the study here we saved time by simply setting up the API access, but perhaps further study could invest time in setting up a more complete dataset.

Further subjects

As mentioned previously, this project may easily expand beyond the wikipedia dataset. A git project history, for instance, may be of interest for further study. We may fairly easily combine the existing research with metrics that concern code in particular, such as Cyclomatic Complexity, which measures code flow complexity according to its logical operators.[31] The main challenge on platforms would be mining the contextual data that was so easily found on Wikipedia.

6.2 Unit testing the CLI

We use the python 'unittest' package to automate most of our testing. The tests can be found in the package root folder in the script 'tester.py' (see appendix A).

Most of the test cases test the argParser.py logic — the module that checks for errors in the CLI arguments — as most of the classes only fail in their tasks given incorrect parameters. We also pay special attention to the return values of each class — in particular the return values on failure. In our package, all exits are intended to be handled by, and filtered up to, the wrhp.py main() function, so these functions must fail gracefully.

6.3 Evaluation of storage system

The database was written initially to take the uniqueness of pageid and revids for granted, but, as the project grew and we began fetching from different language Wikipedias, the database had to change in order to allow for duplicates of both page and revision ID, by adding columns to all tables, and modifying their primary keys.

The database has grew organically with the project in other ways too, and so is un-normalised. It has proven to quite robust and reliable, but as the database grew it became quite slow. We believe it's structure could be improved a little in order to reduce its size.

Figure 6.2 lists the fields and key dependencies of the database. Many of the entities of the database rely on the $(revid, domain)$ key-pair, the only exception being the $Username, Domain \rightarrow UserID$ and $Pageid, Domain \rightarrow Title, Fetchedflag$ relations.

```
P = Pageid
T = Page title
R = Revid
D = Domain
Cn = Content
Un = Username
Ui = Ui
T = Revision timestampx
S = Revision size
Cm = Revision comment
Mw = Math weight
Cw = Citation weight
Fw = Files / Images weight
Lw = Link weight
Sw = Structure weight
Nw = Normal text weight
G = Trajectory gradient
Td = Trajectory distance
F = Fetched flag

RD → PMwCwFwLwSwNwCnCmGSTUnUiTd
PD → TF
UnD → Ui
```

Figure 6.2: Database fields and key dependencies

Comparing this to the database schema as they stand in figure 4.3 we see that some improvements could be made immediately – the $Username, Domain \rightarrow UserID$ relation could receive its own schema immediately, saving a lot of repeated information in the 'wikirevisions' table. This would also be useful as the relation could easily be expanded upon on further study into individual user habits, as I will suggest later.

Similarly, the 'title' attribute of the 'wikirevisions' table could be moved to the 'wikifetched' table, since it is unique to a $(pageid, domain)$ pair. This would necessitate the addition of a boolean column 'fetched' to the same table (the presence or lack of presence of a given $(pageid, domain)$ pair in the 'wikifetched' table currently stands in place of that boolean value). However, given that 'wikifetched' is inevitably much smaller than 'wikirevisions', the space saved would be considerable regardless.

username	domain	userid
⋮	⋮	⋮

(a) Table: wikicontent

pageid	domain	title	fetched?
⋮	⋮	⋮	⋮

(b) Table: wikipages

revid	domain	distance
⋮	⋮	⋮

(c) Table: wikitrajectory

revid	domain	pageid	username	time	size	comment	content	
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

(d) Table: wikirevisions

revid	domain	maths	citations	filesimages	links	structure	normal	gradient
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

(e) Table: wikiweights

Figure 6.3: Recommended new schemata for storing wikipedia data

A further feature that can be done away with is the presence of two revision IDs in the 'wikitrajectory' table. This was useful in an earlier implementations, where the code would manually check for both $(oldrevid, newrevid)$ and $(newrevid, oldrevid)$ pairs before endeavouring to compute a new distance. This feature was scrubbed early but the form of the table has survived. Similarly, content had its own table to facilitate quicker access to revision content before the on-line viewing feature was implemented in the CLI. We could move content to the 'wikirevisions' table, safely take on the convention of the trajectory distance being unique to either the parent or child ID, and reduce the 'wikitrajectory' table by one column.

Some structural inelegance aside, the database is fairly robust. But given the above analysis we should recommend adoption of the schemata described in 6.3. Some separation of figure 6.2's larger relation is recommended, to reduce the space required, whilst keeping the some table separations for performance and maintenance's sake. The large content table is kept separate from the rest of the data, so as to not slow down the database, even though a normalised database would have this data in the wikirevisions table. Maintenance-wise, if one of the distance calculations goes awry, for instance, one would merely have to wipe one table and recalculate.

Since, as demonstrated, we may rely on $(pageid, domain)$ and $(revid, domain)$ pairs, we are able to embed these as native psql primary key restraints. The database will throw error '23000, integrity constraint violation' if this restraint is threatened by the next transaction, which is passed up from the psycopg module as an exception.[44][48] If such an error occurs, we catch it, log it, and terminate the program cleanly.

In reality, though, these errors are infrequent. The code only prepares and inserts a value if it cannot find it in the database, and the insertion functions are implemented as to check before inserting that it will not be duplicating any data.

6.4 Logging

We implement basic but thorough info and debug logging throughout the program, using python's 'logging' package. Our logging functions are defined in 'logger.py' at the root of the package, and by importing the functions there can write detailed info to wikidebug.log. Example output can be found in appendix F.

The CLI also stores details of visited revision in an information log, a excerpt of which can also be found in appendix F.

Otherwise, the package outputs the most useful progress information to the console, including current page, pageid and domain. It also prints dots to the screen during the longer scraping and trajectory and pair-distance calculations. This information is doubled in the debug log, and can be silenced from the console using the '-S' or '-SS' options.

6.5 Machine learning validation attempts

We attempted to validate our model using machine learning. Given the form of the model's output – a 1D array of numbers – machine learning seemed to be a natural choice.

We were interested to see if we could predict gradient factor. Since the gradient factor relates closely to whether or not the edit was included in the final version, we were interested if with machine learning could possibly predict the gradient factor given the weights that we calculate. We could possibly see if the text analysis data we had collected would be able to similarly measure affects on the article.

We used the scikit-learn python package to approach the problem, **??** working with the 180,000 test cases (i.e. complete revision records) extant in the database at the time. We tried predicting gradient factor with the following forms of training data:

- **Weights**
  Simply the calculated weights, as found in the database weight table.
- **Weights and size change**
  The calculated, along with information about whether the revision made the article larger or smaller
- **Weights size change, time change**
  The calculated, along with information about whether the revision made the article larger or smaller
- **Weights summed to one value, and size change**
  Summing the weights to one value is roughly equivalent to taking a plain levenshtein distance of the article on the whole, rather than one separated by species.
- **Summed weights and user edit count over domain**
  We take the generic weight and combine it with the user's activity over the whole wikipedia domain.
- **Summed weights and user edit count over article**
  Similar to above, but only counts user activity in the same article.

We experimented with various types of regression techniques, but our results were not very good, with the regression function rarely fitting to the data with any accuracy. The validation scripts can be run from validator.py, in the validation sub-folder.

Since the gradient factor is sensitive to time, and the scale of that time, we also tried separating the data into two categories, 'towards-final' as 1 and 'away-from-final' as 0, by simply rounding the gradient factor to it's nearest integer, and repeated the same tests as detailed above, but similarly the classification was not very accurate.

We identify two problems that may have effected this.

Our first is perhaps the incompleteness of our data. In particular, the incompleteness of our understanding of the text itself. It was a central premise of this study that we were to make no attempt at understanding the actual content of the text.

But perhaps the actual fact content of the text is quite important, particularly on an encyclopaedia, to its survival — perhaps Wikipedia is predictable in this regard, that facts have an innate discoverable quality. Perhaps in this case we would be in a better place to predict success if we could measure the factual accuracy of new text, run a spell checks on it, etc.

Our second problem may be that, perhaps, much of the data we analysed actually doesn't greatly effect the survival rate of the edit. Much of the information lost from and gained by pages like Rupert Sheldrake's, or Derek Smart's was factually correct, and not necessarily malformed. The reason for the exclusion of content was to quell a conflict in the case of the former text, whereas in the latter page it was the personal opinions of individual Wikipedia editors about which facts actually were facts that caused (and continue to cause) complications.

Talk page data is one thing, and we looked briefly at how the talk page data can correlate with article data, but external context is relevant too. Rupert Sheldrake's interview on the BBC coincides with changes in his article, and we can refer to Lih's 2004 article on edits made immediately after celebrity deaths.

We note that there are less regular editors on Wikipedia now than there was in 2007, user-count 'has shrunk by more than a third since [then] and is still shrinking'.[56] Wikipedia's 'formal mechanisms for normal articulation are shown to have calcified against changes – especially changes proposed by newer editors'.[19] The study identifies, amongst other things, a number of tools, bots and policies that have often tended to revert entries regardless of their content.

But Wikipedia's internal problems are of not real issue with this study, they only give us reason to doubt whether we could use a simple dataset in order to characterise the complex conditions that conduce a successful contribution to Wikipedia.

Rather, the lesson we learn here is simply that a wide context is necessary for understanding and characterising real-world human collaboration.

Chapter 7

Conclusion

7.1 Applications

The software, as it stands, provides a fairly open and malleable source for examining the histories of Wikipedia articles. As a history-visualisation tool, it is different to existing tools and allows flexibility in analysing different kinds of pages.

The modularity of the approach also allows for more radical changes to the software. The project could be easily adapted to handle different data, with little change. To handle GitHub document history, for instance, merely changing the database wrapper and API wrapper classes could be enough to begin analysing the data there, substituting the page identifier for a file identifier, revision identifier for commit. Our classes generally only need the two-part unique identification in order to proceed with analysis.

7.2 Summary

Wikipedia proved to be a good case study for the exploring how we may automatically analyse an individual's collaborative stake in a joint, text based work.

We were able to analyse the greater context of each article, as well as the content text. It is perhaps one of the most interesting parts of Wiki analysis that evidence of interaction is so readily available. Due to the nature of the WikiMedia software, these interactions are as publicly available as the article's themselves, all utilising the same text revision framework. Through this we were able to expand our knowledge of the circumstance of the collaboration, and by proxy our understanding of each collaborative act.

Our conclusions are ambiguous. As this work stands, building upon the work of the previous year to achieve an automatic and expandable framework for analysis, it makes a healthy attempt at describing each edit in terms of it's input. We may even describe our weighted levenshtein distance calculator to be a lossy compression algorithm, reducing the text down to its meaningful parts.

However, our second measure, gradient factor, which describes both the local and long-term success of a revision did not seem to correlate well with the above.

We acknowledge that naïve string analysis, by definition, excludes the recognition of quality content in terms of facts, well-formedness, and so on, so perhaps our measure is incomplete. These qualities affect the result of a collaborative act, of course, particularly when contributing to a knowledge repository. But we also acknowledge that the correlation between edit content and edit success may not have a simple correlation.

For all the studies that sought out to analyse objective quality in articles, and edits, there are also studies of Wikipedia's biased attitudes to what edits may stay in an article. The part of our study that concerned talk-pages and arbitration requests contributed to the latter line of enquiry.

And we believe that this may not merely be a problem native to Wikipedia, or on-line discourse. We believe that Wikipedia may merely present a more-than-usually analysable trace of the inter-personal relations that shape all collaborations. In some of the cases described here these inter-actions are evidenced in talk-page communication. In others, like in undo-redo edit wars, the interaction is embedded in the act of editing itself.

And this is not to mention the myriad external factors that also affect regard to individual edits – long-standing off-Wikipedia arguments in both the Rupert Sheldrake and Derek Smart articles referred to previously are good examples. Again, with search engines we have the opportunity to track these waves of opinion – complex changes in the standard of knowledge of that occur naturally.

But the individual predilections of key, more-active editors, if not evidenced in previous edits, remain a mystery. As may, of course, the habits and opinions of a manger in a group-coding scenario, until they are codified into action.

The major lesson we learn overall, then, is that if we can't predict the fore-coming trajectory of a work, we may certainly have the means to analyse it once the fact of the history of the work is crystallised into its final form. We achieved this by deciding the article was conceptually 'finished' at the most recent revision we fetch. The 'finished' state, however, occurs naturally in other forms of work.

If surrounding context as important as it seems in this study, then we have learnt that a user's stake in a collaborative work should be as affected by the degree of discussion and coordination that that individual has engaged in as much as it is by the remaining artefacts of his or her contributions extant in the work's final form.

In the chaos of on-line interaction that characterises Wikipedia, we must look not only to the final work. We must take into account measures of interaction, and a users weight in the overall community network of the on-line space. Their valuable contributions are not only textual, but social; the collaboration is not merely the sum of its parts, but also product of a community, a network, and a society.

Bibliography

[1] Steffen Abel and Athanasios Theologis. "Odyssey of auxin". In: *Cold Spring Harbor perspectives in biology* 2.10 (2010), a004572 (see p. 36).

[2] *AcaWiki*. `http://acawiki.org`. Accessed: 2014-04-31 (see p. 9).

[3] Tim Adams. *Rupert Sheldrake: the 'heretic' at odds with scientific dogma*. [Online; accessed 2014-08-28]. 2012. URL: `http://www.theguardian.com/science/2012/feb/05/rupert-sheldrake-interview-science-delusion` (see p. 36).

[4] B Thomas Adler. "WikiTrust: content-driven reputation for the Wikipedia". In: (2012) (see p. 2).

[5] B. Thomas Adler and Luca de Alfaro. "A Content-driven Reputation System for the Wikipedia". In: *Proceedings of the 16th International Conference on World Wide Web*. WWW '07. Banff, Alberta, Canada: ACM, 2007, pp. 261–270. ISBN: 978-1-59593-654-7. DOI: `10.1145/1242572.1242608`. URL: `http://doi.acm.org/10.1145/1242572.1242608` (see pp. 9, 13).

[6] Luluah Al-Husain and Abdulrahman Mirza. "A Survey of Online Collaboration Tools". In: (2010) (see p. 1).

[7] *Alexa About Us*. `http://www.alexa.com/about`. Accessed: 2014-04-31 (see p. 8).

[8] Joshua E Blumenstock. "Size matters: word count as a measure of quality on wikipedia". In: *Proceedings of the 17th international conference on World Wide Web*. ACM. 2008, pp. 1095–1096 (see p. 9).

[9] William I. Chang and Jordan Lampe. "Theoretical and Empirical Comparisons of Approximate String Matching Algorithms". In: *Proceedings of the Third Annual Symposium on Combinatorial Pattern Matching*. CPM '92. London, UK, UK: Springer-Verlag, 1992, pp. 175–184. ISBN: 3-540-56024-6. URL: `http://dl.acm.org/citation.cfm?id=647812.738270` (see p. 7).

[10] Ed H Chi, Bongwon Suh, and Aniket Kittur. "Providing social transparency through visualizations in Wikipedia". In: *Proceedings of the Social Data Analysis Workshop at CHI*. Vol. 8. 2008 (see p. 16).

[11] Tom Cross. "Puppy smoothies: Improving the reliability of open, collaborative wikis". In: *First Monday* 11.9 (2006). ISSN: 13960466. URL: http://firstmonday.org/ojs/index.php/fm/article/view/1400 (see p. 9).

[12] Luca De Alfaro, Ashutosh Kulshreshtha, Ian Pye, and B. Thomas Adler. "Reputation Systems for Open Collaboration". In: *Commun. ACM* 54.8 (Aug. 2011), pp. 81–87. ISSN: 0001-0782. DOI: 10.1145/1978542.1978560. URL: http://doi.acm.org/10.1145/1978542.1978560 (see p. 9).

[13] Peter Denning, Jim Horning, David Parnas, and Lauren Weinstein. "Wikipedia Risks". In: *Commun. ACM* 48.12 (Dec. 2005), pp. 152–152. ISSN: 0001-0782. DOI: 10.1145/1101779.1101804. URL: http://doi.acm.org/10.1145/1101779.1101804 (see p. 18).

[14] Various developers. *Google Code — JWPL and the Wikipedia Revision Toolkit*. [Online; accessed 2014-06-03]. 2012. URL: https://code.google.com/p/jwpl/ (see p. 23).

[15] Various developers. *Wikipedia Miner*. [Online; accessed 2014-06-03]. 2012. URL: http://wikipedia-miner.cms.waikato.ac.nz/services/ (see p. 23).

[16] Dave Evans. "The Workplace of The Future: Connected, Collaborative, Creative". In: (2013) (see p. 1).

[17] Oliver Ferschke, Torsten Zesch, and Iryna Gurevych. "Wikipedia Revision Toolkit: Efficiently Accessing Wikipedia's Edit History". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Systems Demonstrations*. HLT '11. Portland, Oregon: Association for Computational Linguistics, 2011, pp. 97–102. ISBN: 978-1-932432-90-9. URL: http://dl.acm.org/citation.cfm?id=2002440.2002457 (see p. 23).

[18] Jim Giles. "Internet encyclopaedias go head to head". In: *Nature* 438.7070 (2005), pp. 900–901. ISSN: 0028-0836. URL: http://dx.doi.org/10.1038/438900a (see pp. 18, 42).

[19] Aaron Halfaker, R. Stuart Geiger, Jonathan Morgan, and John Riedl. "The Rise and Decline of an Open Collaboration System: How Wikipedia's reaction to sudden popularity is causing its decline". In: *American Behavioral Scientist* 57.5 (2013), pp. 664–688. DOI: 10.1177/0002764212469365. URL: http://dx.doi.org/10.1177/0002764212469365 (see p. 46).

[20] Heikki Hyyrö. "A bit-vector algorithm for computing Levenshtein and Damerau edit distances". In: *Nordic Journal of Computing* (2003), p. 2003 (see p. 7).

[21] Takashi Iba, Keiichi Nemoto, Bernd Peters, and Peter A. Gloor. "Analyzing the Creative Editing Behavior of Wikipedia Editors: Through Dynamic Social Network Analysis". In: *Procedia - Social and Behavioral Sciences* 2.4 (2010). The 1st Collaborative Innovation Networks Conference - {COINs2009}, pp. 6441 –6456. ISSN: 1877-0428. DOI: http://dx.doi.org/10.1016/j.sbspro.2010.04.054. URL: http://www.sciencedirect.com/science/article/pii/S1877042810011122 (see p. 10).

[22] Aniket Kittur, Ed H. Chi, and Bongwon Suh. "What's in Wikipedia?: Mapping Topics and Conflict Using Socially Annotated Category Structure". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '09. Boston, MA, USA: ACM, 2009, pp. 1509–1512. ISBN: 978-1-60558-246-7. DOI: 10.1145/1518701.1518930. URL: http://doi.acm.org/10.1145/1518701.1518930 (see p. 10).

[23] Aniket Kittur and Robert E. Kraut. "Beyond Wikipedia: Coordination and Conflict in Online Production Groups". In: *Proceedings of the 2010 ACM Conference on Computer Supported Cooperative Work*. CSCW '10. Savannah, Georgia, USA: ACM, 2010, pp. 215–224. ISBN: 978-1-60558-795-0. DOI: 10.1145/1718918.1718959. URL: http://doi.acm.org/10.1145/1718918.1718959 (see p. 10).

[24] Aniket Kittur, Bongwon Suh, Bryan A. Pendleton, and Ed H. Chi. "He Says, She Says: Conflict and Coordination in Wikipedia". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '07. San Jose, California, USA: ACM, 2007, pp. 453–462. ISBN: 978-1-59593-593-9. DOI: 10.1145/1240624.1240698. URL: http://doi.acm.org/10.1145/1240624.1240698 (see p. 10).

[25] VI Levenshtein. "Binary Codes Capable of Correcting Deletions, Insertions and Reversals". In: *Soviet Physics Doklady* 10 (1966), p. 707 (see p. 5).

[26] Michael Lieberman and Jimmy Lin. *You Are Where You Edit: Locating Wikipedia Contributors through Edit Histories*. 2009. URL: http://aaai.org/ocs/index.php/ICWSM/09/paper/view/204 (see p. 16).

[27] Andrew Lih. "Wikipedia as Participatory journalism: reliable sources? metrics for evaluating collaborative media as a news resource". In: *In Proceedings of the 5th International Symposium on Online Journalism*. 2004, pp. 16–17 (see p. 10).

[28] Teun Lucassen and Jan Schraagen. "Evaluating WikiTrust: A trust support tool for Wikipedia". In: *First Monday* 16.5 (2011). ISSN: 13960466. URL: http://firstmonday.org/ojs/index.php/fm/article/view/3070 (see pp. 2, 9).

[29] Teun Lucassen and Jan Maarten Schraagen. "Trust in Wikipedia: How Users Trust Information from an Unknown Source". In: *Proceedings of the 4th Workshop on Information Credibility*. WICOW '10. Raleigh, North Carolina, USA: ACM, 2010, pp. 19–26. ISBN: 978-1-60558-940-4. DOI: 10.1145/1772938.1772944. URL: http://doi.acm.org/10.1145/1772938.1772944 (see p. 9).

[30] matplotlib. *matplotlib: python plotting*. [Online; accessed 2014-08-28]. 2014. URL: http://matplotlib.org/ (see p. 23).

[31] Thomas J. McCabe. "A Complexity Measure". In: *Proceedings of the 2Nd International Conference on Software Engineering*. ICSE '76. San Francisco, California, USA: IEEE Computer Society Press, 1976, pp. 407–. URL: http://dl.acm.org/citation.cfm?id=800253.807712 (see p. 42).

[32] Deborah L McGuinness, Honglei Zeng, Paulo Pinheiro Da Silva, Li Ding, Dhyanesh Narayanan, and Mayukh Bhaowal. "Investigations into Trust for Collaborative Information Repositories: A Wikipedia Case Study." In: *MTW* 190 (2006) (see p. 9).

[33] MediaWiki. *API:Main page*. [Online; accessed 2014-08-22]. 2014. URL: http://www.mediawiki.org/w/index.php?title=API:Main_page&oldid=1113489 (see p. 11).

[34] Mostafa Mesgari, Chitu Okoli, Mohamad Mehdi, Finn Årup Nielsen, and Arto Lanamäki. ""The sum of all human knowledge": A systematic review of scholarly research on the content of Wikipedia". In: *Journal of the American Society for Information Science and Technology* (2014). This is a postprint of an article accepted for publication in Journal of the American Society for Information Science and Technology copyright © 2014 (American Society for Information Science and Technology). URL: http://spectrum.library.concordia.ca/978618/ (see pp. 8, 55).

[35] Mårton Mestyån, Taha Yasseri, and Jånos Kertåsz. "Early Prediction of Movie Box Office Success based on Wikipedia Activity Big Data". In: *CoRR* abs/1211.0970 (2012). URL: http://dblp.uni-trier.de/db/journals/corr/corr1211.html#abs-1211-0970 (see p. 16).

[36] Gene Myers. "A Fast Bit-vector Algorithm for Approximate String Matching Based on Dynamic Programming". In: *J. ACM* 46.3 (May 1999), pp. 395–415. ISSN: 0004-5411. DOI: 10.1145/316542.316550. URL: http://doi.acm.org/10.1145/316542.316550 (see p. 7).

[37] Daniel Nasaw. *BBC News — Meet the 'bots' that edit Wikipedia*. [Online; accessed 2014-06-02]. 2012. URL: http://en.wikipedia.org/w/index.php?title=Wikipedia:Bots&oldid=602889799 (see p. 16).

[38] The Proffessional Copywriter's Network. *Recommended rates for hiring copywriters*. 2014. URL: http://www.procopywriters.co.uk/recommended-rates-for-hiring-copywriters/ (visited on 05/20/2014) (see p. 4).

[39] Intellectual Property Office. *Joint authors*. 2014. URL: http://www.ipo.gov.uk/types/copy/c-ownership/c-jointauthors.htm (visited on 05/20/2014) (see pp. 3, 4).

[40] UK Intellectual Property Office. *Intellectual property - an overview*. 2014. URL: https://www.gov.uk/intellectual-property-an-overview/overview (visited on 05/20/2014) (see p. 3).

[41] Roderic Page. "Visualising edit history of a Wikipedia page". In: (2009). URL: http://iphylo.blogspot.co.uk/2009/09/visualising-edit-history-of-wikipedia.html (visited on 07/15/2014) (see p. 16).

[42] Roderic Page. *Wiki History Flow*. 2011. URL: https://github.com/rdmpage/wikihistoryflow (visited on 07/15/2014) (see p. 16).

[43] Lydia Pintscher. *Wikidata live on the English Wikipedia*. [Online; accessed 2014-08-25. 2013. URL: http://blog.wikimedia.de/2013/02/13/wikidata-live-on-the-english-wikipedia/ (see p. 33).

[44] PostgreSQL. *PostgreSQL documentation: Error codes*. [Online; accessed 2014-08-28]. 2014. URL: http://www.postgresql.org/docs/8.3/static/errcodes-appendix.html (see p. 44).

[45] PostgreSQL. *PostgreSQL documentation: TOAST*. [Online; accessed 2014-08-27]. 2014. URL: http://www.postgresql.org/docs/current/interactive/storage-toast.html (see p. 23).

[46] Martin Potthast, Benno Stein, and Robert Gerling. "Automatic Vandalism Detection in Wikipedia". In: *Advances in Information Retrieval*. Ed. by Craig Macdonald, Iadh Ounis, Vassilis Plachouras, Ian Ruthven, and RyenW. White. Vol. 4956. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, pp. 663–668. ISBN: 978-3-540-78645-0. DOI: 10.1007/978-3-540-78646-7_75. URL: http://dx.doi.org/10.1007/978-3-540-78646-7_75 (see p. 10).

[47] psycopg2. *PostgreSQL + Python — Psycopg*. [Online; accessed 2014-08-28]. 2014. URL: http://initd.org/psycopg/ (see pp. 23, 25).

[48] psycopg2. *psycopg2 documentation: Error codes*. [Online; accessed 2014-08-28]. 2014. URL: http://initd.org/psycopg/docs/errorcodes.html (see p. 44).

[49] *Python Extending Python with C or C++*. https://docs.python.org/2/extending/extending.html. Accessed: 2014-08-29 (see p. 23).

[50] *Python wikipedia 1.3.0*. https://pypi.python.org/pypi/wikipedia. "Wikipedia is a Python library that makes it easy to access and parse data from Wikipedia." Accessed: 2014-04-31 (see p. 23).

[51] Ruth Raitman, Naomi Augar, and Wanlei Zhou. "Employing wikis for online collaboration in the e-learning environment: Case study". In: *Information Technology and Applications, 2005. ICITA 2005. Third International Conference on*. Vol. 2. IEEE. 2005, pp. 142–146 (see p. 1).

[52] Mikalai Sabel. "Structuring Wiki Revision History". In: *Proceedings of the 2007 International Symposium on Wikis*. WikiSym '07. Montreal, Quebec, Canada: ACM, 2007, pp. 125–130. ISBN: 978-1-59593-861-9. DOI: 10.1145/1296951.1296965. URL: http://doi.acm.org/10.1145/1296951.1296965 (see p. 16).

[53] Damien Salauze. "A Simple Method For Calculating A" Fair" Royalty Rate". In: (2011) (see p. 4).

[54] Rupert Sheldrake. *Rupert Sheldrake: autobiography*. [Online; accessed 2014-08-28]. 2005. URL: http://www.sheldrake.org/about-rupert-sheldrake/autobiography (see p. 36).

[55] Rupert Sheldrake and Dan Damon. *World Service: World Update*. Radio interview. 2013 (see p. 38).

[56] Tom Simonite. "The Decline of Wikipedia". In: (2013). URL: http://www.technologyreview.com/featuredstory/520446/the-decline-of-wikipedia/ (see p. 46).

[57] Temple F Smith and Michael S Waterman. "Identification of common molecular subsequences". In: *Journal of molecular biology* 147.1 (1981), pp. 195–197 (see p. 7).

[58] T.F. Smith and M.S. Waterman. "Identification of common molecular subsequences". In: *Journal of Molecular Biology* 147.1 (1981), pp. 195 –197. ISSN: 0022-2836. DOI: http://dx.doi.org/10.1016/0022-2836(81)90087-5. URL: http://www.sciencedirect.com/science/article/pii/0022283681900875 (see p. 7).

[59] B. Stvilia, M.B. Twidale, L.C. Smith, and L. Gasser. "Assessing information quality of a community-based encyclopedia". In: *Proceedings of the International Conference on Information Quality*. 2005, pp. 442–454 (see p. 9).

[60] B. Suh, Ed H. Chi, B.A. Pendleton, and A. Kittur. "Us vs. Them: Understanding Social Dynamics in Wikipedia with Revert Graph Visualizations". In: *Visual Analytics Science and Technology, 2007. VAST 2007. IEEE Symposium on*. 2007, pp. 163–170. DOI: 10.1109/VAST.2007.4389010 (see p. 16).

[61] Bongwon Suh, Gregorio Convertino, Ed H. Chi, and Peter Pirolli. "The Singularity is Not Near: Slowing Growth of Wikipedia". In: *Proceedings of the 5th International Symposium on Wikis and Open Collaboration*. WikiSym '09. Orlando, Florida: ACM, 2009, 8:1–8:10. ISBN: 978-1-60558-730-1. DOI: 10.1145/1641309.1641322. URL: http://doi.acm.org/10.1145/1641309.1641322 (see p. 10).

[62] Walter F. Tichy. "The String-to-string Correction Problem with Block Moves". In: *ACM Trans. Comput. Syst.* 2.4 (Nov. 1984), pp. 309–321. ISSN: 0734-2071. DOI: 10.1145/357401.357404. URL: http://doi.acm.org/10.1145/357401.357404 (see p. 42).

[63] Stanford University Libraries: Copyright & Fair Use. *Copyright Ownership: Who Owns What?* 2014. URL: http://fairuse.stanford.edu/overview/faqs/copyright-ownership/ (visited on 05/20/2014) (see p. 3).

[64] 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*. CHI '04. Vienna, Austria: ACM, 2004, pp. 575–582. ISBN: 1-58113-702-8. DOI: 10.1145/985692.985765. URL: http://doi.acm.org/10.1145/985692.985765 (see p. 16).

[65] Rome Viharo. *Wikipedia, we have a problem*. 2014. URL: http://wikipediawehaveaproblem.com/ (visited on 08/03/2014) (see p. 38).

[66] Jimmy Wales. *WikiEN-l Mailing List: Re: Copyright situation*. 2005. URL: https://lists.wikimedia.org/pipermail/wikien-l/2005-August/027373.html (visited on 05/20/2014) (see p. 3).

[67] *WikiLit*. http://wikilit.referata.com. A temporary site, relating directly to [34]. Accessed: 2014-04-31 (see p. 9).

[68] *Wikimedia enwiki dump progress on 20140502*. http://dumps.wikimedia.org/enwiki/20140502/. Accessed: 2014-04-29 (see p. 42).

[69] *Wikimedia Wikipedia.org is more popular than...: Note on Alexa rankings*. https://meta.wikimedia.org/wiki/Wikipedia.org_is_more_popular_than...#Note_on_Alexa_rankings. Accessed: 2014-04-31 (see p. 8).

[70] *WikiPapers*. http://wikipapers.referata.com. Accessed: 2014-04-31 (see p. 9).

[71] Wikipedia. *Consensus*. 2014. URL: http://en.wikipedia.org/wiki/Wikipedia:Consensus (visited on 05/20/2014) (see p. 4).

[72] Wikipedia. *FAQ/Overview:Who is responsible for the articles on Wikipedia*. 2014. URL: http://en.wikipedia.org/wiki/Wikipedia:FAQ/Overview#Who_is_responsible_for_the_articles_on_Wikipedia.3F (visited on 05/20/2014) (see p. 4).

[73] Wikipedia. *Help: Reverting — Wikipedia, The Free Encyclopedia*. [Online; accessed 2014-06-02]. 2014. URL: http://en.wikipedia.org/w/index.php?title=Help:Reverting&oldid=606567776 (see p. 14).

[74] Wikipedia. *Wikipedia:Bot policy — Wikipedia, The Free Encyclopedia*. [Online; accessed 2014-08-25]. URL: http://en.wikipedia.org/w/index.php?title=Wikipedia:Bot_policy&oldid=620647443 (see p. 33).

[75] Wikipedia. *Wikipedia:Bots — Wikipedia, The Free Encyclopedia*. [Online; accessed 2014-06-02]. 2014. URL: http://en.wikipedia.org/w/index.php?title=Wikipedia:Bots&oldid=602889799 (see p. 16).

[76] Wikipedia. *Wikipedia:Copyrights*. 2014. URL: http://en.wikipedia.org/wiki/Wikipedia:Copyrights (visited on 06/14/2014) (see p. 1).

[77] Wikipedia. *Wikipedia:Good articles*. 2014. URL: http://en.wikipedia.org/wiki/Wikipedia:Good_articles (visited on 05/20/2014) (see p. 4).

[78] Wikipedia. *Wikipedia:Wikidata*. [Online; accessed 2014-08-25. 2014. URL: http://en.wikipedia.org/w/index.php?title=Wikipedia:Wikidata&oldid=619979765 (see p. 33).

[79] Dennis M. Wilkinson and Bernardo A. Huberman. "Assessing the Value of Coooperation in Wikipedia". In: *CoRR* abs/cs/0702140 (2007) (see p. 8).

[80] Jianmin Wu and Mizuho Iwaihara. "Revision Graph Extraction in Wikipedia Based on Supergram Decomposition". In: *Proceedings of the 9th International Symposium on Open Collaboration*. WikiSym '13. Hong Kong, China: ACM, 2013, 10:1–10:6. ISBN: 978-1-4503-1852-5. DOI: 10.1145/2491055.2491065. URL: http://doi.acm.org/10.1145/2491055.2491065 (see p. 16).

[81] Honglei Zeng, Maher A. Alhossaini, 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: Bridge the Gap Between PST Technologies and Business Services*. PST '06. Markham, Ontario, Canada: ACM, 2006, 8:1–8:1. ISBN: 1-59593-604-1. DOI: 10.1145/1501434.1501445. URL: http://doi.acm.org/10.1145/1501434.1501445 (see p. 9).

Appendix A

Guide to code base

```
/wikiInterface
├── __init__.py
├── consts.py
├── wrhp.py   ◁ the base CLI main()
├── argParser.py
├── wikiDataHandle.py
├── wikiDataPlot.py
├── wikiAnalysis.py   ◁ analytical methods
├── launch.py   ◁ opens wikipedia pages in default browser
├── logger.py   ◁ shared logging functions
├── info.log   ◁ info log destination
├── debug.log   ◁ debug log destination
├── unittests.py   ◁ unit testing scripts
├── dbpas
├── lshtein   ◁ the levenshtein calculator files
│   ├── __init__.py
│   ├── fastlev.so
│   └── fastlev
│       ├── fastlev.cpp
│       ├── setup.py
│       └── Makefile
├── scraper
│   ├── wikiRevisionScrape.py
│   ├── __init__.py
│   └── langs.csv   ◁ details of different language wikipedias
├── database
│   ├── __init__.py
│   ├── wikiData.py   ◁ database wrapper
│   └── schemamake.sql   ◁ database schema definitions
└── validate   ◁ the validation / testing scripts
    └── validator.py
```

Appendix B

Edit and share plots selection

P.T.O.

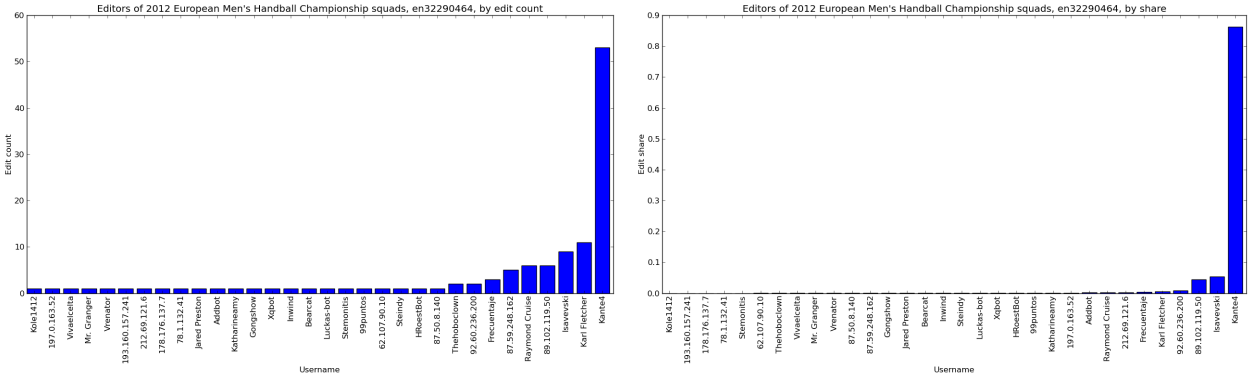


Figure B.1: English Wikipedia, Page ID 32290464 (European Men's Handball Championship squads)

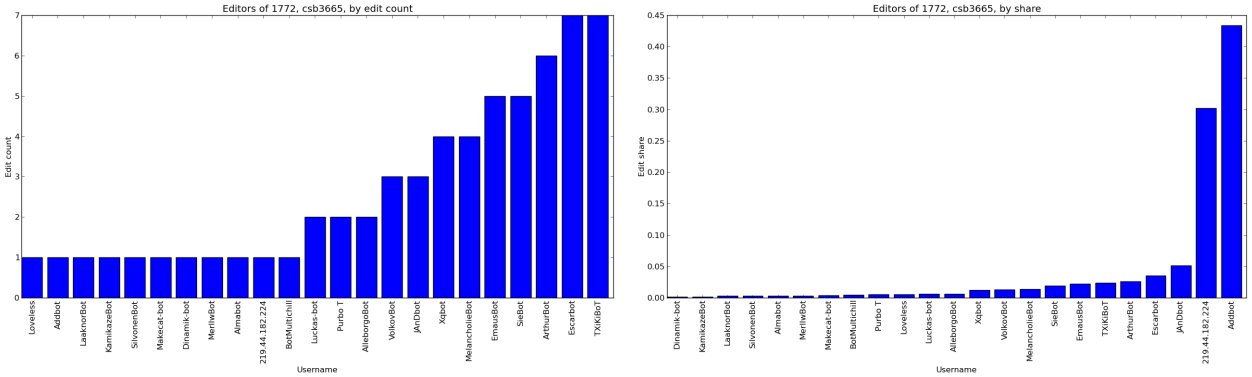


Figure B.2: Kashubian Wikipedia, Page ID 3665 (1772)

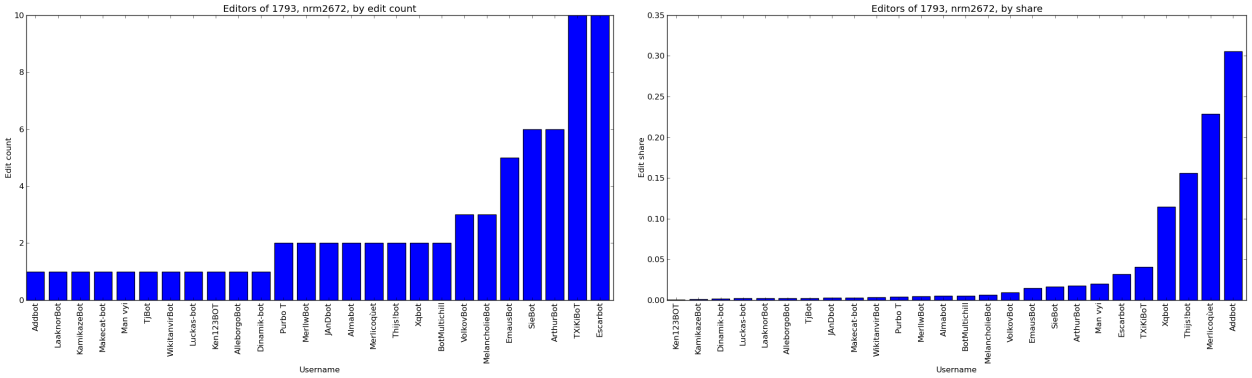


Figure B.3: Norman Wikipedia, Page ID 2672 (1793)

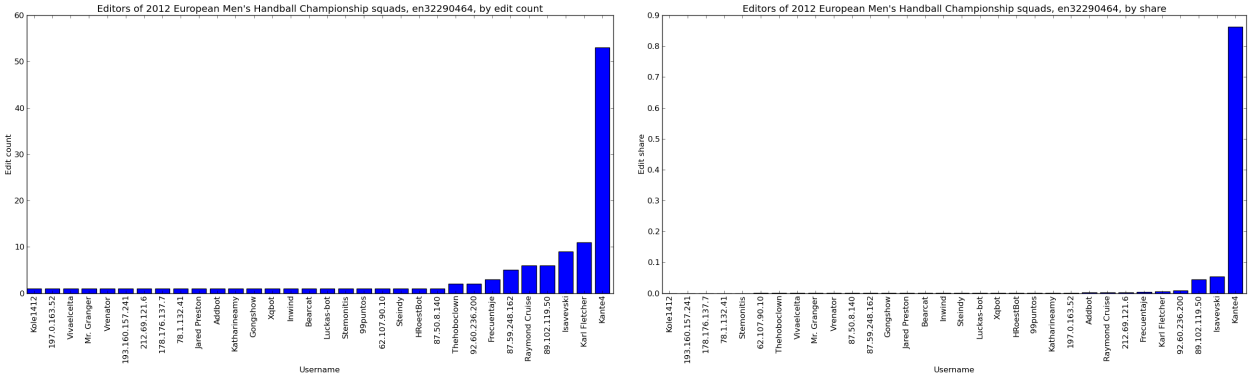


Figure B.4: English Wikipedia, Page ID 32290464 (European Men's Handball Championship squads)

Appendix C

Combination plot selection

P.T.O.

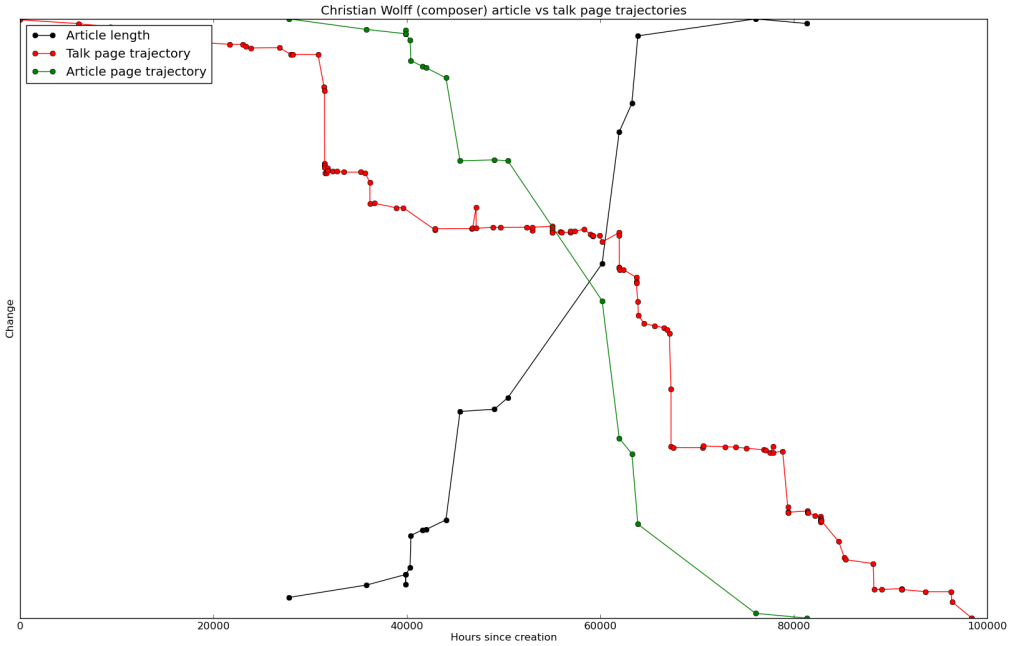


Figure C.1: English page ID 241976, (Christian Wolff (composer))

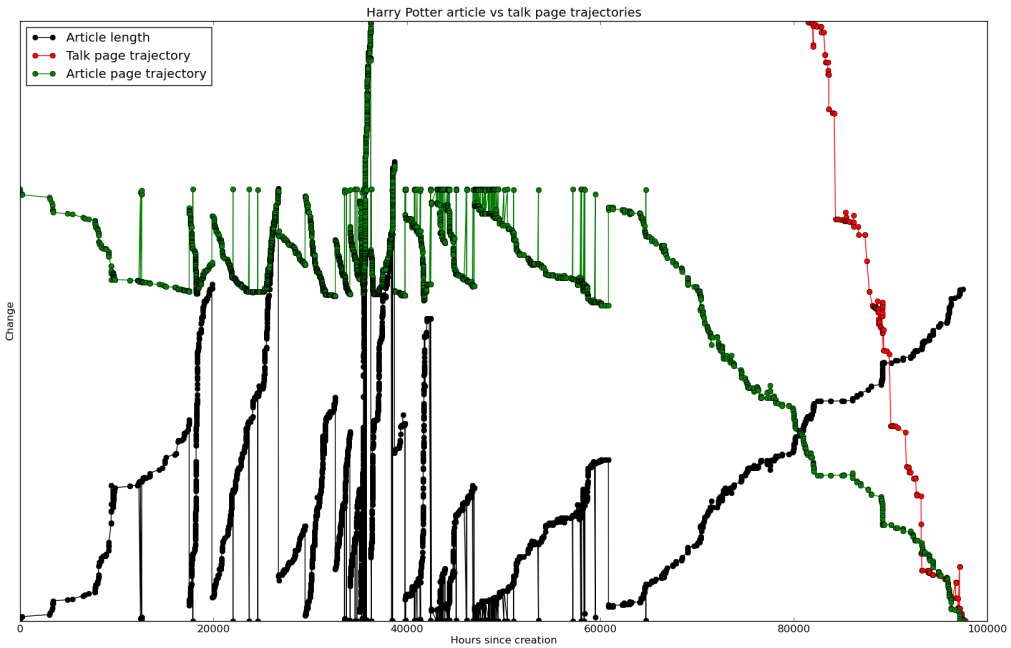


Figure C.2: English Wikipedia, page ID 2387806 (Harry Potter)

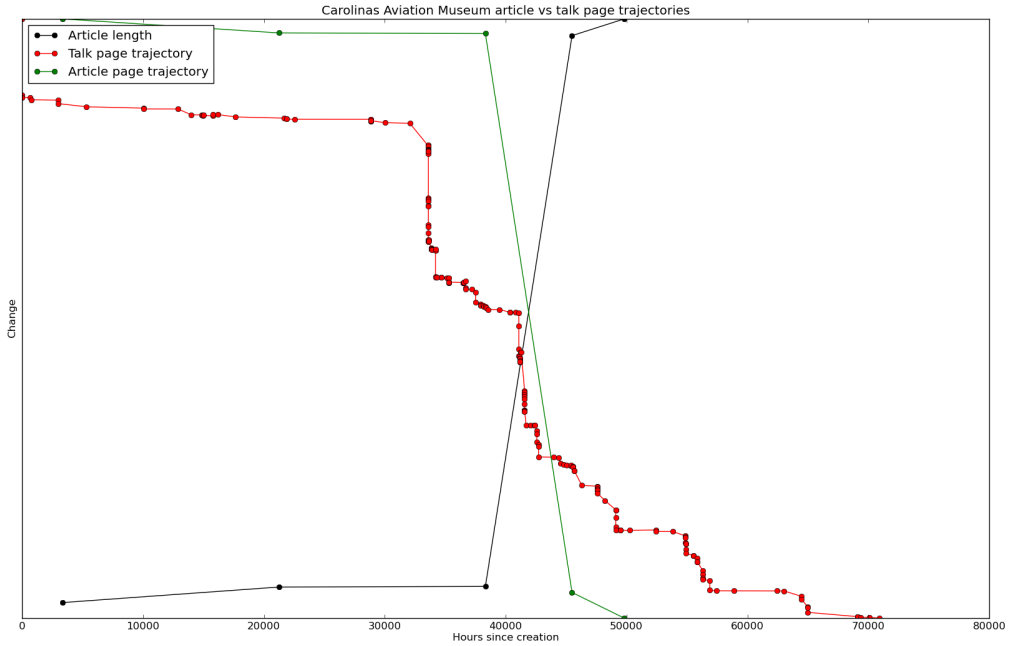


Figure C.3: English Wikipedia, page ID 4834368 (Carolinas Aviation Museum)

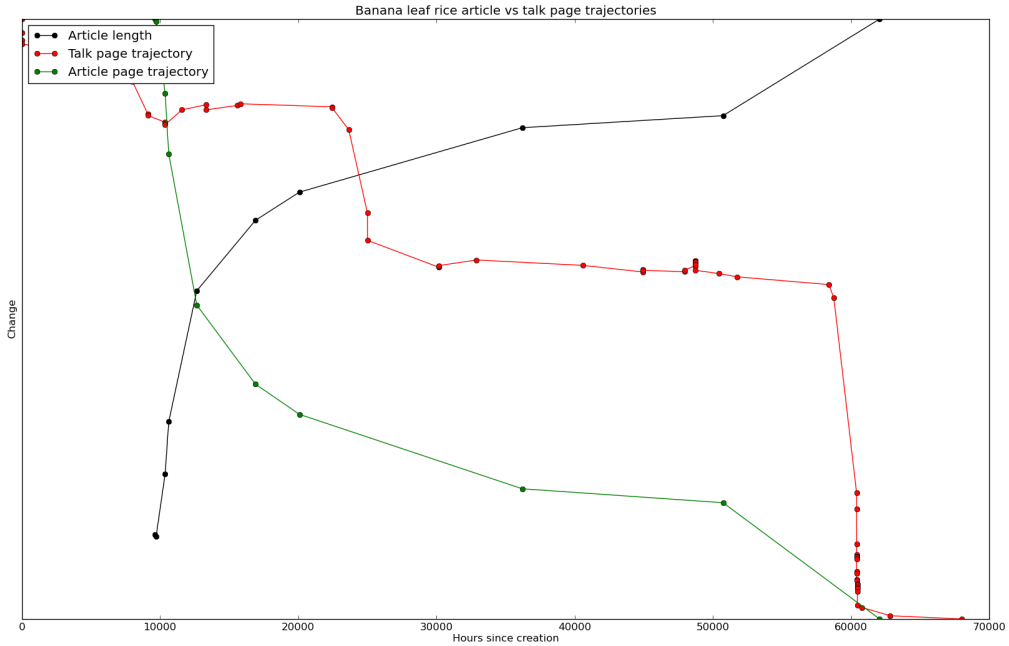


Figure C.4: English Wikipedia page ID 5800180 (Banana leaf rice)

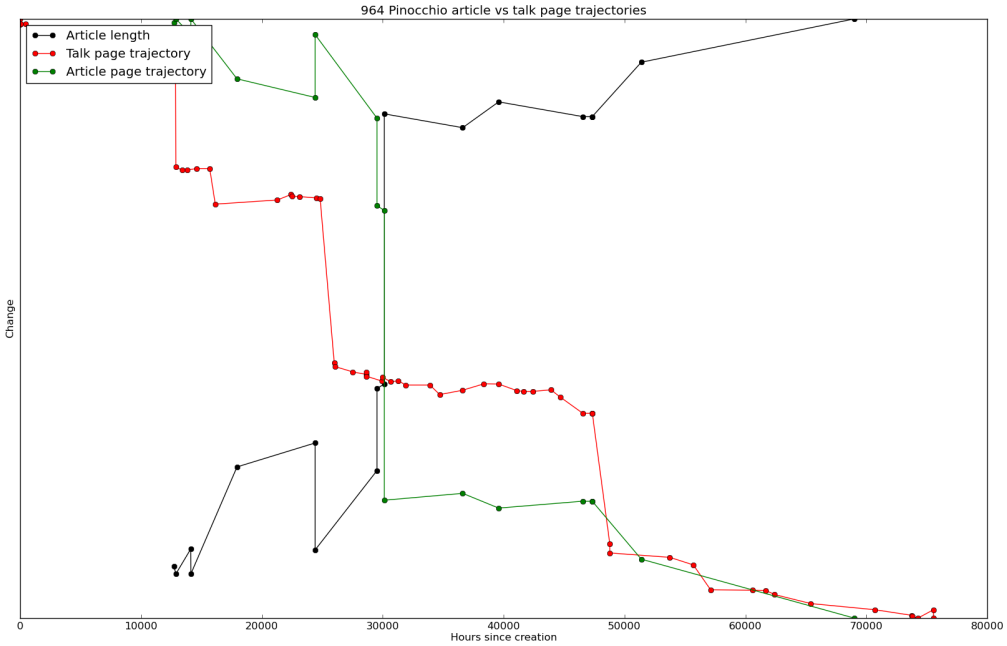


Figure C.5: English Wikipedia, page ID 9497885 (964 Pinocchio)

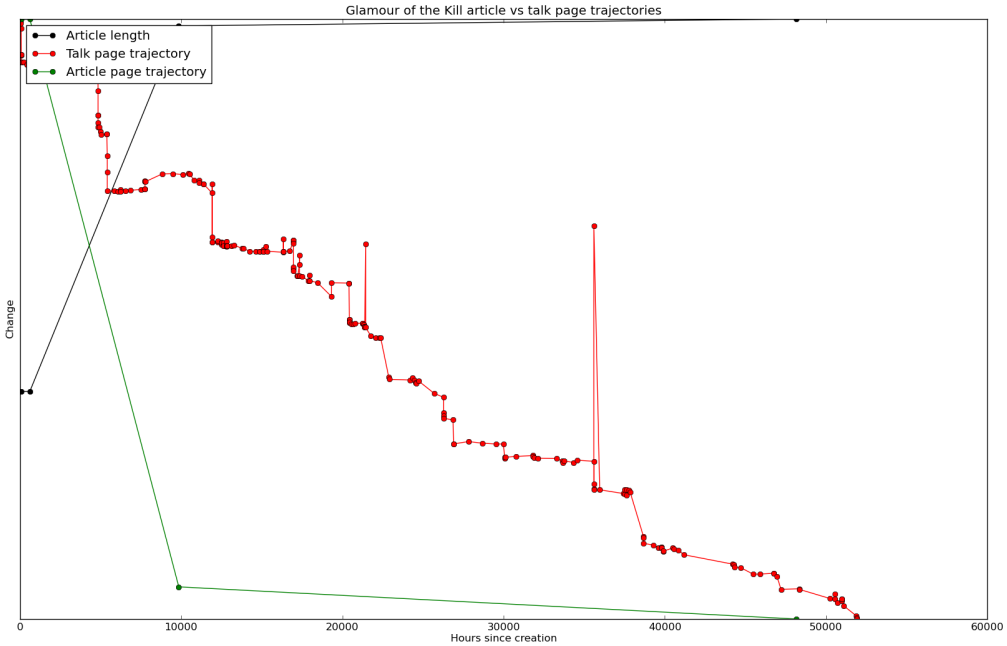


Figure C.6: English Wikipedia, page ID 18190882 (Glamour of the Kill)

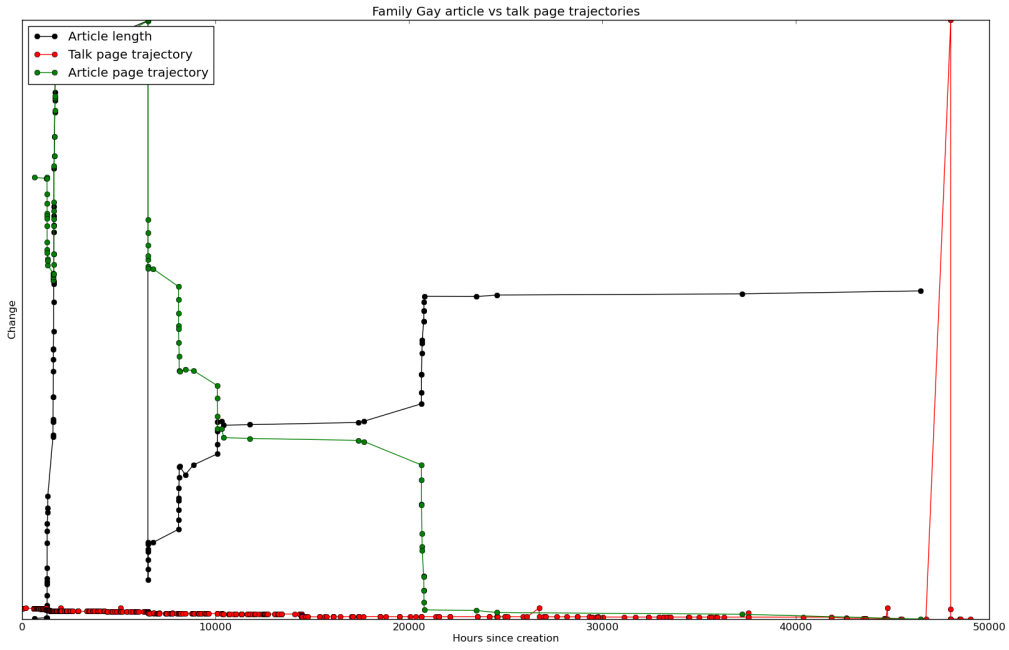


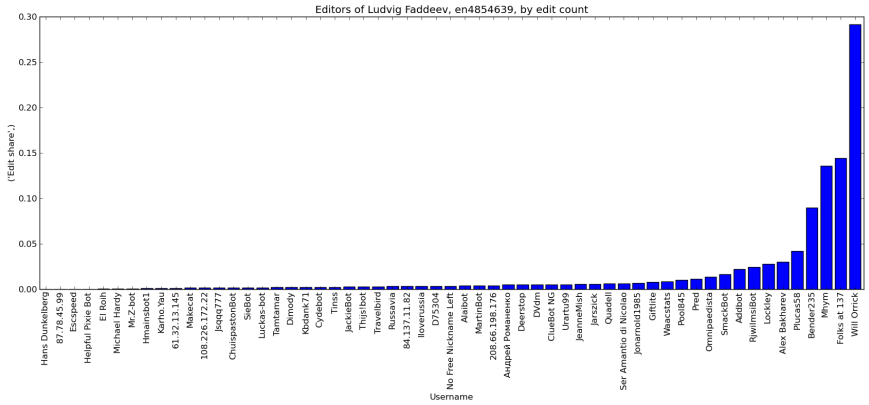
Figure C.7: English Wikipedia, page ID 21113485 (Family Gay)

Appendix D

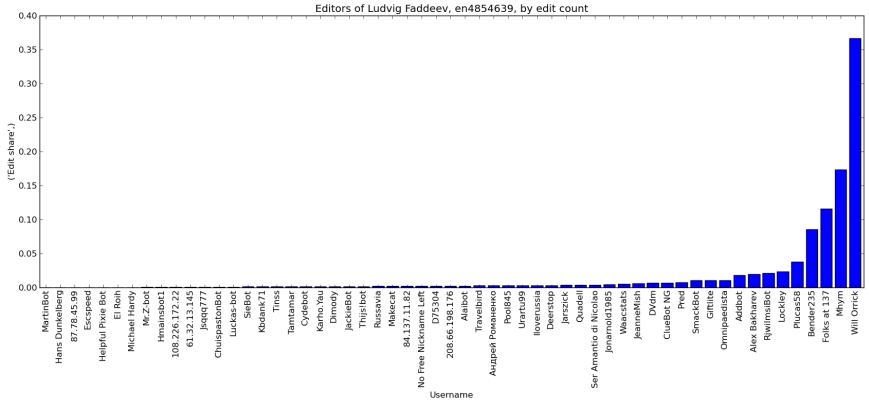
Weighted share graphs

P.T.O.

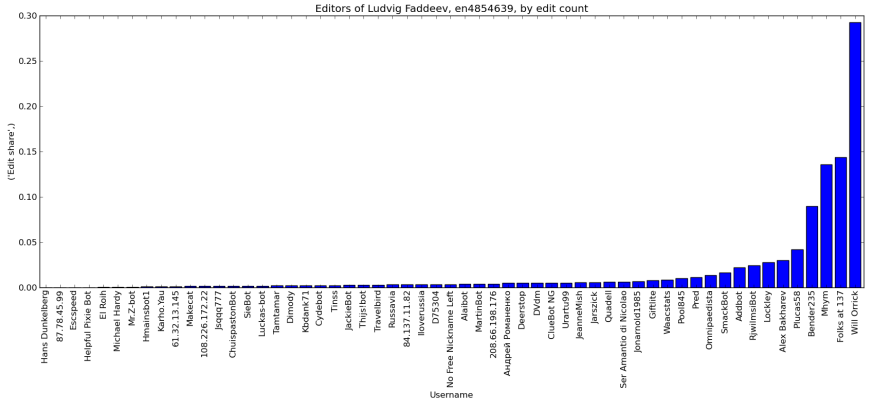
Figure D.1: English Wikipedia, Page ID 4854639 (Ludvig Faddeev)



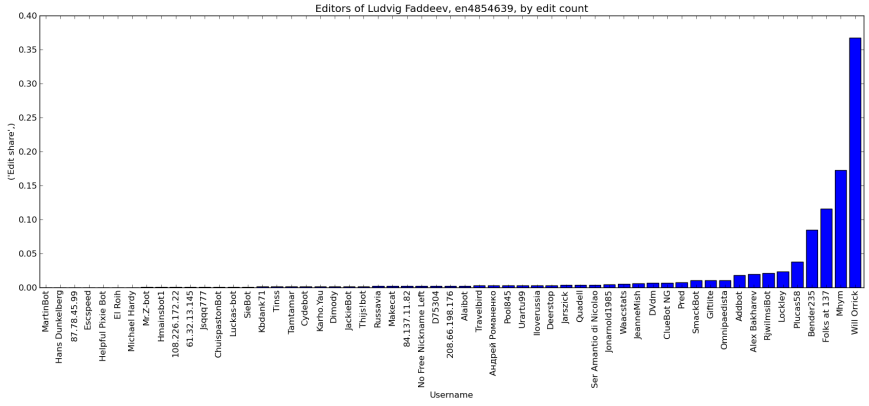
(a) Unweighted



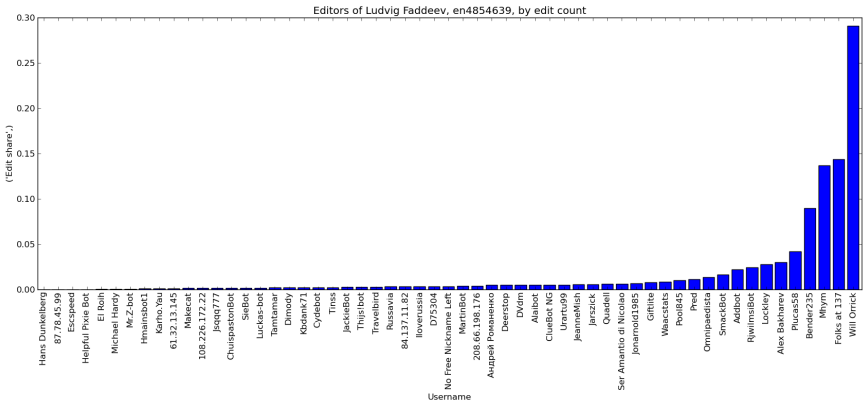
(b) Unweighted with gradient



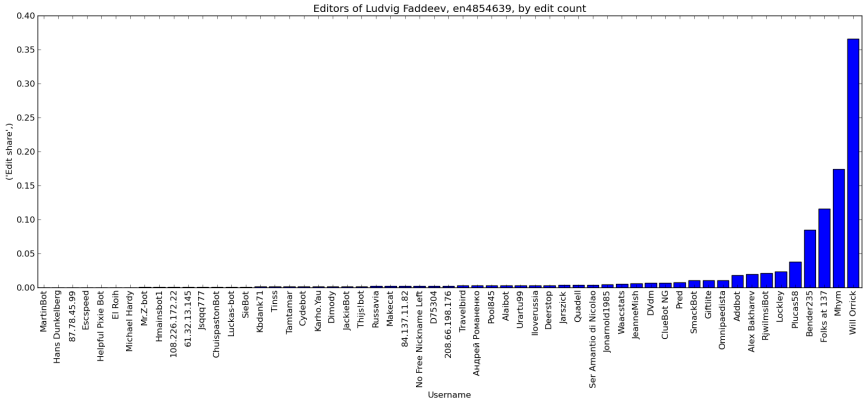
(c) ...with double structure weight



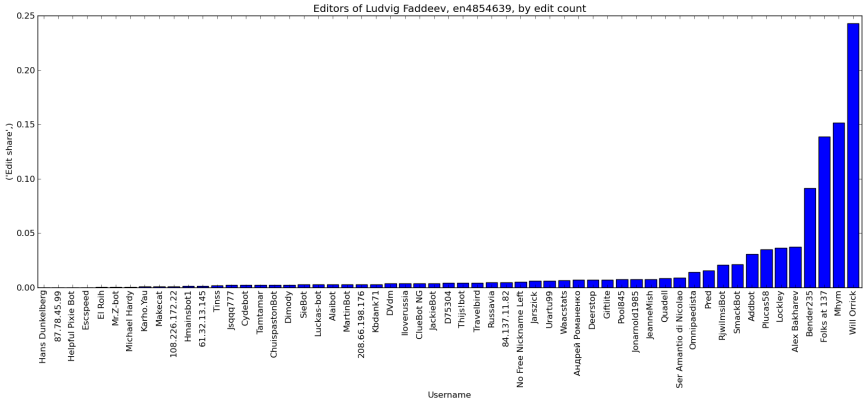
(d) ..plus gradient factor



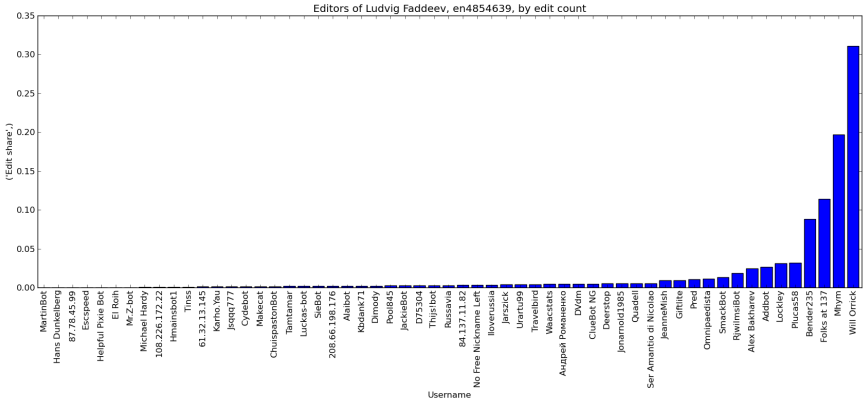
(e) ...with double maths weight



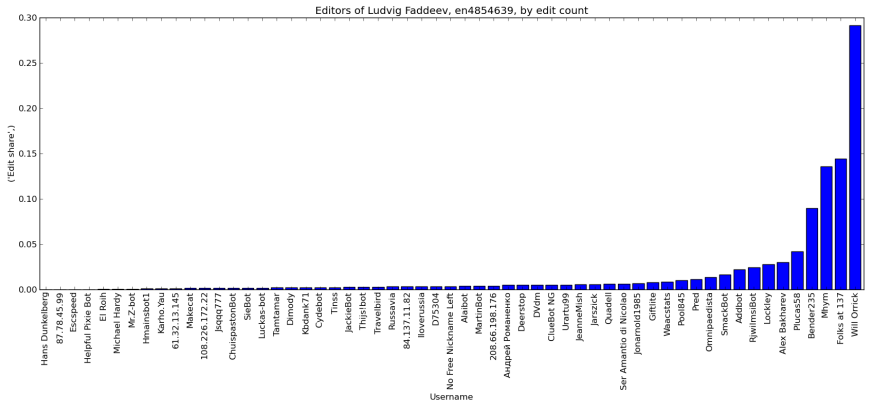
(f) ..plus gradient factor



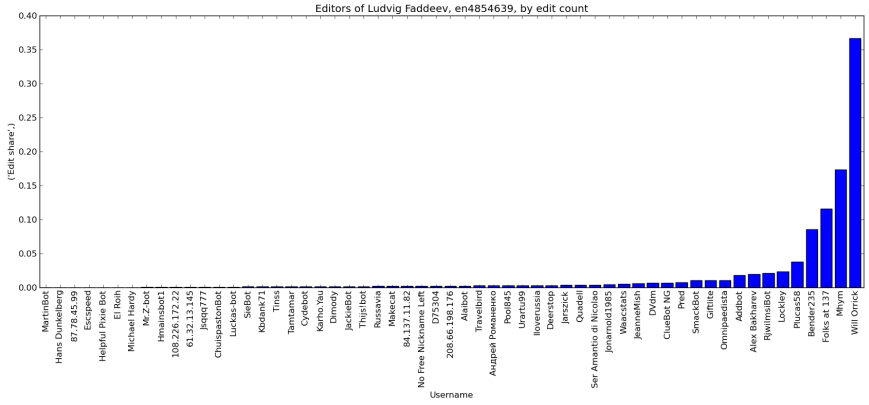
(g) ...with double link weight



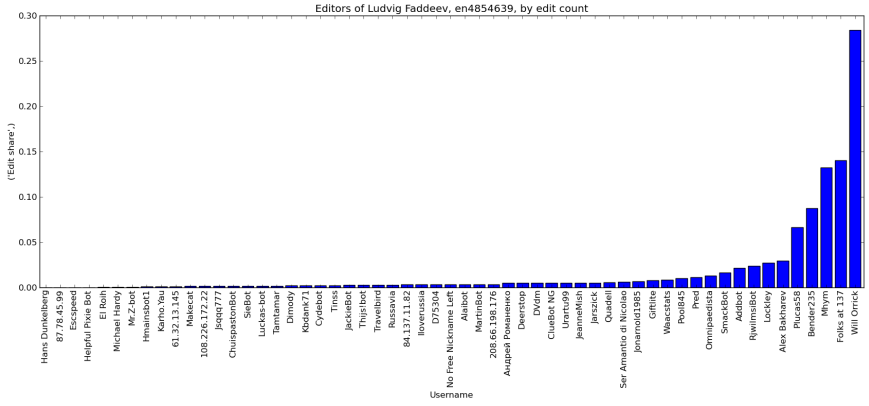
(h) ..plus gradient factor



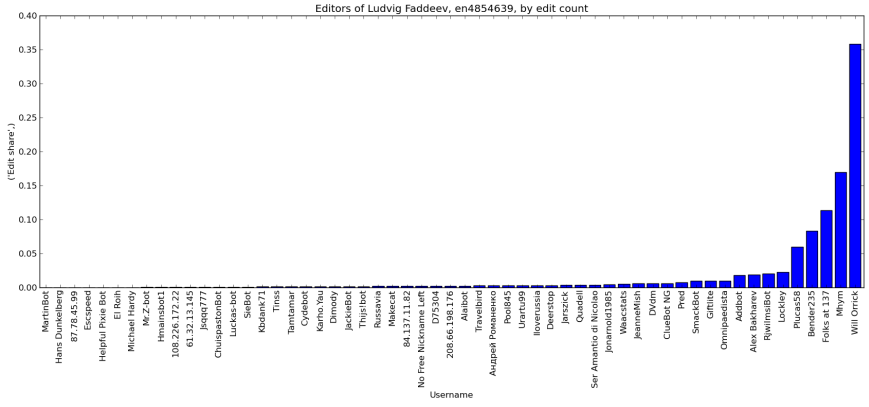
(a) ...with double files / images weight



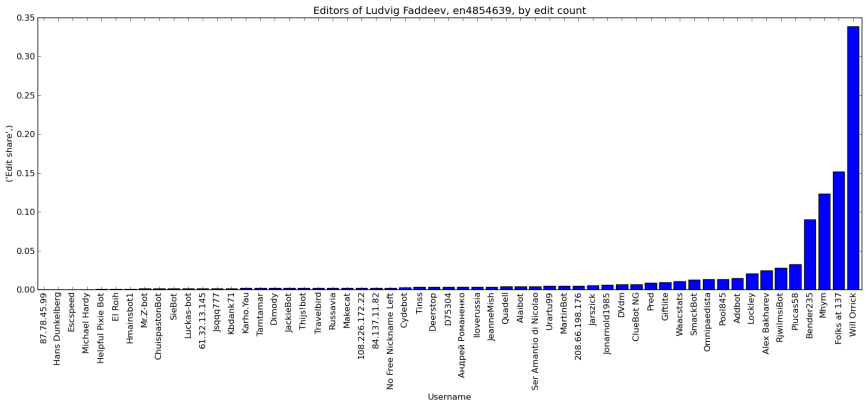
(b) ..plus gradient factor



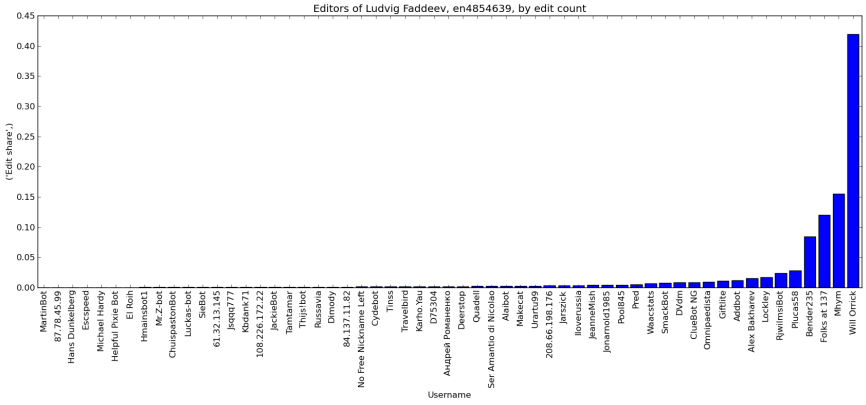
(c) ...with double citations weight



(d) ..plus gradient factor



(a) ...with double normal weight



(b) ..plus gradient factor

Appendix E

SQL snippets, and results

This appendix is reffered to in the evaluation section.

E.1 Average discrepancy as a percentage of overall diff

```
   SELECT AVG(
@(@(w.maths +
w.citations +
w.filesimages +
w.links +
w.structure +
w.normal)
- @(t1.distance -
t2.distance))
/ CASE WHEN
CASE WHEN
@(w.maths +
w.citations +
w.filesimages +
w.links +
w.structure +
w.normal) > @(t1.distance -
t2.distance)
THEN
@(w.maths +
w.citations +
w.filesimages +
w.links +
w.structure +
w.normal)::float
ELSE
@(t1.distance -
t2.distance)::float
END
= 0
THEN 1
ELSE
CASE WHEN
@(w.maths +
w.citations +
w.filesimages +
w.links +
w.structure +
w.normal) > @(t1.distance -
t2.distance)
THEN
@(w.maths +
w.citations +
w.filesimages +
w.links +
w.structure +
w.normal)::float
ELSE
@(t1.distance -
t2.distance)::float
END
```

```
   END) AS average_discrepancy

       FROM wikitrajectory AS t1
       JOIN wikiweights AS w
       ON t1.revid2 = w.revid
       AND t1.domain = w.domain

       JOIN wikirevisions AS r
       ON r.revid = w.revid
       AND r.domain=w.domain

       JOIN wikitrajectory AS t2
       ON r.parentid = t2.revid2
       AND r.domain = t2.domain;

 average_discrepancy_percent
-----------------------------
      0.386195332179081
(1 row)

(END)
```

E.2 Matching diffs vs non matching diffs

```
wm613=> SELECT SUM(
   CASE WHEN
         @(@(w.maths +
       w.citations +
          w.filesimages +
       w.links +
       w.structure +
       w.normal)
         - @(t1.distance-
       t2.distance)) = 0
         THEN 1
   ELSE 0
   END) AS match,
       SUM(
         CASE WHEN
   @(@(w.maths +
   w.citations +
   w.filesimages +
   w.links +
         w.structure +
   w.normal) -
   @(t1.distance-
   t2.distance)) <> 0
   THEN 1
   ELSE 0
   END) AS non_match

       FROM wikitrajectory AS t1
       JOIN wikiweights AS w
       ON t1.revid2 = w.revid
       AND t1.domain = w.domain

       JOIN wikirevisions AS r
       ON r.revid = w.revid
       AND r.domain=w.domain

       JOIN wikitrajectory AS t2
       ON r.parentid = t2.revid2
       AND r.domain = t2.domain;

 match | non_match
-------+-----------
 30836 |    72250
(1 row)

(END)
```

E.3 Average diff mismatch

```
      SELECT AVG(
        @(@(w.maths +
    w.citations +
    w.filesimages +
    w.links +
    w.structure +
    w.normal)
    - @(t1.distance
      - t2.distance)) /
    CASE WHEN
    r.size = 0
    THEN
    1
    ELSE
    r.size::float) AS average_discrepency

    FROM wikitrajectory AS t1
    JOIN wikiweights AS w
    ON t1.revid2 = w.revid
    AND t1.domain = w.domain

    JOIN wikirevisions AS r
    ON r.revid = w.revid
    AND r.domain=w.domain

    JOIN wikitrajectory AS t2
    ON r.parentid = t2.revid2
    AND r.domain = t2.domain;

 average_discrepency
---------------------
    22180.7321410815
(1 row)

(END)
```

Appendix F

Log examples

F.1 Debug log excerpt

Showing the reporting of spawned processes, their workload, and their return time.

```
2014-09-04 17:57:43,594 analyse:('getting weights, pair parentid:',
                                                 49618798, 'childid:', 52943690)
2014-09-04 17:57:43,596 analyse:('in database',)
2014-09-04 17:57:43,597 analyse:('getting weights, pair parentid:',
                                                 0, 'childid:', 49618798)
2014-09-04 17:57:43,706 calcdists:('spawned', 2)
2014-09-04 17:57:43,709 calcdistworker:('worker process 5 string1
                                               length: 0 string2 length: 130',)
2014-09-04 17:57:43,712 calcdistworker:('process 5 returning 130',)
2014-09-04 17:57:43,711 calcdistworker:('process 3 returning 69',)
2014-09-04 17:57:43,715 calcdists:('recieved', 2, 'results')
2014-09-04 17:57:43,715 makeweight:('calculated gradient as 1',)
analyse:('in database',)
```

F.2 Info log excerpt

The end of a report on the weight and gradient factor calculations made for each revision. The bottom entry shows the reporting of the above calculation.

```
-----[69639596] / 2006-08-14 19:14:11-----
[0, 0, 0, 10, 0, 17, 0.503424]
 TOTAL 13.592448

-----[67378284] / 2006-08-03 04:23:27-----
[15, 0, 0, 0, 0, 29, 0.514538]
 TOTAL 22.639672

-----[65177530] / 2006-07-22 07:56:50-----
[0, 0, 0, 57, 0, 44, 0.504509]
 TOTAL 50.955409

-----[55399013] / 2006-05-27 10:51:49-----
[0, 0, 0, 17, 0, 0, 0.999917]
 TOTAL 16.998589

-----[55398992] / 2006-05-27 10:51:33-----
[0, 0, 0, 17, 0, 1, 0.500929]
 TOTAL 9.016722
```

```
-----[52951347] / 2006-05-13 04:12:12-----
[0, 0, 0, 2, 0, 0, 0.968231]
 TOTAL 1.936462

-----[52949980] / 2006-05-13 04:00:11-----
[0, 0, 0, 72, 0, 104, 0.99954]
 TOTAL 175.91904

-----[52948570] / 2006-05-13 03:46:24-----
[15, 0, 0, 789, 0, 571, 0.999799]
 TOTAL 1374.723625

-----[52943690] / 2006-05-13 03:02:53-----
[0, 0, 0, 0, 0, 0, 0.5]
 TOTAL 0.0

-----[49618798] / 2006-04-22 17:29:01-----
[0, 0, 0, 69, 0, 130, 1.0]
 TOTAL 199.0

END----------------------------------
```