Copyright Model for Collaboration

Literature Review

William Marsey
Imperial College London

June 2014

1 Introduction

1.1 The Research Question

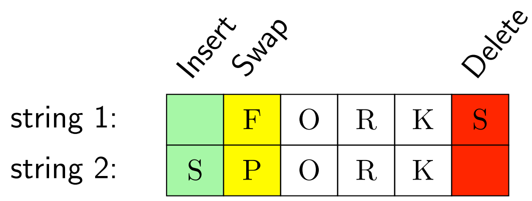
Given a collaboratively edited document, and information about the collection of edits that constitute that document, may we measure the quality of each contribution? And may we use that to give all the collaborators a algorithmically-defined 'stake' in that final document?

Collaborative work is becoming a big deal. It is both interesting and an important trend in modern computer use. And the data is abundant.

We begin with a brief overview of the history of the edit distance problem and similar issues, including the carious algorithms that may be used. We then look at the particulars of Wikipedia, and the academic studies that are most related to what we are developing here.

1.2 Edit difference

To measure difference between different text revisions, we will refer to edit distance. Edit distance between two texts, as defined in the research initiated by Levenshtein in 1966,[14] can be defined as the minimum amount of insert, delete and substitutions operations needed to transform one text into another.



forks → spork, edit distance: 3

Figure 1: An edit distance example using all three operations

Levenshtein's characterisation of this distance is given as:

That is, the distance between two strings is characterised the minimum distance between three different pair-combinations of its substrings. A 'text-book' implementation of this algorithm can be represented by the pseudo-code below. (We present the dynamic-programming-style algorithm here, and will generally be working with dynamic programming implementations throughout the study.)

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

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

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

```
ed(x,y):
  #end base cases
  if |x| = 0: return |y|
  if |y| = 0: return |x|

  #end table initialisation
  d is a table [0..|x|][0..|y|]
  for i = 1 to |m|:
     d[i,0] = i
  for j = 0 to |y|:
     d[0,j] = j

  #dynamic computation
  for j = 1 to |y|:
     for i = 1 to |x|:
        c = [(x[i] == y[j]) ? 0 else 1]
        ins = d[i-1,j] + 1
        dlt =d[i,j-1] + 1
        kp_swp = d[i-1,j-1] + c
        d[i,j] = min(ins, dlt, kp-swp)

  #return last computed number
  return d[|x|,|y|]
```

Figure 2: Basic dynamic implementation of Levenshtein distance

The algorithm runs in $\theta(|x||y|)$ time, with x and y being the two strings being compared — we can clearly see the derivation of this bound from the creation of the $|x|$ by $|y|$. For the same reason the space complexity of the algorithm is also $\theta(|x||y|)$.

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 $d[i,j]$ (as it appears above), we see that we require only part of the matrix: $d[i-1,j-1]$, $d[i-1,j]$ and $d[i,j-1]$. Depending on the implementation, we may at any point decide to either disregard rows $0 \ldots i-2$ inclusive, or columns $0 \ldots j-2$ (where $i-2$ or $j-2 > 0$, respectively).

There are more complicated techniques for disregarding unneccesary computations — a few implementations employ strategies that allow them to trace the table space diagonally, tracing a rather than iteratively, achieving a time complexities as low as $O(ed(x,y)^2)$.[5] Another harnesses bit vectors 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.[22]

Extensions can also be made to the nature of the distance itself. Work on such additions, adapting the generic edit distance to a variety of different and specific needs. Here is a brief overview of the main groups these extensions fall into:

- **Hamming distance.** This allows for substitutions only, comparing same-length strings, such that:
  $ed_{hamming}(\text{“abc”}, \text{“abd”}) = 1$,
  $ed_{hamming}(\text{“abc”}, \text{“bcd”}) = 3$,
  and $ed_{hamming}(\text{“abc”}, \text{“ab”})$ is undefined.[10]
- **Reversals.** The Damerau-Levenshtein distance defines an extra operation, which is the swap of to adjacent characters. It is particularly suited to spell-checking and for analysing DNA-sequence variations. In this case:
  $ed_{damerau}(\text{“ab”}, \text{“ba”}) = 1$
- **Block distance.** This allows for displacements of entire blocks to count as one operation.
  $ed_{block}(\text{“abcde”}, \text{“cdeax”}) = 2$
  (one move of the block ‘cde’, one substitution of ‘b’ for ‘x’)[29]
- ***q*-grams distance.** q-grams are simply sub-strings, and this measure describes the similarity of two strings in terms of q-grams they share.[**Ukkonen1992**] This variations is quite different from the other algorithms, while remaining comparable:
  $ed_{q-gram}(x, y) = \sum_{v \in \Sigma^q} |G(x)[v] - G(y)[v]|$
  where $G(x)[v]$ returns the number of occurrences of q-gram v in string x, and Σ^q is all the possible q-grams in the alphabet (capped by string length). $|G(x)[v] - G(y)[v]|$ a large positive number every time a q-gram appears a large amount of times in one string, but not the other; it returns 0 if the substring apears the same number of times. So, the whole function measures this difference for all possible substrings, and sums them, returning a high number for difference, and a low number for similarity.

Other algorithms we may look to are those that concern themselves with common subsequences. The common subsequence problem relates to the edit distance problem by way of the heuristic that two similar strings will have similar subsequences — the q-gram algorithm just mentioned relies on this heuristic, and works well for most texts, it does not work for all measures. For example, two strings that are very different according to this heuristic may be quite similar according to the Damrau-Levenshtein measure.

Another part of the problem of working out optimal edit distance is calculating the optimal alignment — the measures are colesly related. For example, in figure 1, the alignment of the two strings “fork” and “spork” was:

[H]
s	p	o	r	k	-
-	f	o	r	k	s

However it could have conceivably also been:

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

We can see how the edit distance for the right-hand example would be sub-optimal given this alignment.

The Smith-Waterman algorithm[26] calculates optimal alignment by populating two tables — one like the one in the pseudocode above, as well as a table of arrows. These arrows define a path from one corner of the table space to the other. The shape of this path defines how to align the two strings.

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

- A $\nwarrow$ at $[i, j]$, if $x[i] \neq y[j]$, denotes a ’swap’ between $x[i]$ and $y[j]$ (otherwise they are the lack of an operation).
- A $\uparrow$ at $[i, j]$ denotes the deletion of $x[i]$
- A $\leftarrow$ at $[i, j]$ denotes the insertion of $y[j]$

$$\left\{\begin{array}{ccccccc}
 & & S & P & O & R & K \\
 & {\color{red}0} & {\color{red}0} & 0 & 0 & 0 & 0 \\
F & 0 & \nwarrow & {\color{red}\nwarrow} & \nwarrow & \nwarrow & \nwarrow \\
O & 0 & \uparrow & \nwarrow & {\color{red}\nwarrow} & \downarrow & \leftarrow \\
R & 0 & \uparrow & \uparrow & \uparrow & {\color{red}\nwarrow} & \leftarrow \\
K & 0 & \uparrow & \uparrow & \uparrow & \uparrow & {\color{red}\nwarrow} \\
S & 0 & \nwarrow & \uparrow & \uparrow & \uparrow & {\color{red}\uparrow}
\end{array}\right\}$$

(If the arrow 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.)

Figure 3: Diagram showing Smith-Waterman traceback (in red) on the edit operation forks $\rightarrow$ spork

bit vector implementation[12]

Finally, we may also look into Delta algorithms. These describe ways of constructing Delta encodings of a document's history — a compression format in which only the difference between each text is stored, not the entire version. These algorithms are of the same family of algorithms discussed above. In fact, we find that one of the fastest known algorithms, VDelta, is a refinement of the block distance algorithm mentioned above.[1] For a given version n of a document doc is given by:

$$v_n = v_0 \cup \Delta(v_0, v_1) \cup \Delta(v_1, v_2) \cup \cdots \cup \Delta(v_{n-1}, v_n)$$

where $\Delta(v_i, v_j)$ is the difference between version i and version j of the document, and the $\cup$ operation combines each version in a manner particular to the Δ data-type. Storing data in this way can be very efficient, resulting in a compression factors of five or ten on typical data.[19] It may also be relatively easy to maintain in our case because of the linear nature of our versions.

2 Previous work

2.1 Wikipedia

Wikipedia's pre-eminence as an online resource is self-evident to anyone who has searched the internet for a generic topic. The website is ranked 6$^{\text{th}}$ globally in terms of website traffic,[2] and is the highest-ranked reference website by far - most of the sites it shares the top spots with are portals, search engines, shopping mega-sites, and social media websites.[4] Despite some skepticism (particularly concern over the inherent chaos of the system: "...edits, contributed in a predominantly undirected and haphazard fashion by ... unvetted volunteers."[39]), 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'[20].

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 — an attestment to the observation that Wikipedia has been 'irresistable point of unquiry for researchers from various fields of knowledge', and a useful touching stone for this study. Mesgari et al's review finds 82 out of the 100 to concern quality in Wikipedia articles,

[1] In Hunt's 1998 study[11]

[2] According to Alexa, an Amazon-owned company. The statistics are wide-rangingbased on a combined measure of Unique Visitors and Pageviews, and the data mined from around 25,000 different browser extensions, as well as sites that have installed Alexa's scripts.[3] Alexa may well be biased towards English speakers and Internet Explorer users, but this may underestimate Wikipedia.org's popularity, since 'two thirds of all Wikipedia articles are in languages other than English'[33]

some of these are also referenced here, and many of the others will come to bear on the study as it progresses.

Other important general sources will be WikiLit,[31] AcaWiki[1] and WikiPapers[34], all of which are online repositories of academic research into Wikipedia and other Wikis (as well as being Wikis themselves...).

The six 'risks' one takes when referencing Wikipedia, as defined in an early article on the subject,[8] is a good starting block for identifying the ways in which to ragard the 'quality' of content in Wikipedia. We list them here, describing the implications for our work with each. Some are particular to Wikipedia, some are inherent to all Wikis.

- **Accuracy.** It is important to remember that, without severely increasingly the complexity of our algorithm, we may not verify the accuracy of information. And, if accuracy of information is proportionate to value (surely it must be in a reference text), then our algorithm may misplace its reward. We may most usefully look at the problem as follows. The texts that are edited most often are those that are visited most often. The previously cited studies of Lih and Mestyan et al attest to this - they both study the peaks in activity in articles that are brought to attention in some way. We find in the work of Bongwon Suh et al that the growth of wikipedia is inverse-exponential, as the overheads of coordination and beaurocrosy temper content creation.[28][13] Content is more likely to be refined and corrected as an article grows old.[39] We can assume, then, that all articles tend towards accuracy (we may find this bore out in Giles's 2005 semi-formal comparison of Encyclopedia Britannica articles to Wikipedia articles - finding an average three mistakes in the former and 4 mistakes in the latter)[9]. We may possibly extend this to say that all edits improve a text. This is complicated by malicious, misinformed or malformed edits, but we will discuss dealing with these later.
  Response:
- **Motives.** It has been found that different contributors may edit Wikipedia in various different ways, according to their proclivities.
  Response:
- **Uncertain Expertise.** We may take this to mean malformed and misinformed editing, but we may also take it to mean malicious editing. As for malicious edits - we find a lot of useful information in Potthast et al's work on automatic detection of vandals,[24] as well as the discussions around Wikipedia's own Counter-Vandalism Unit ('CVU').[37] including
- **Volatility.**
- **Coverage.** Cite that structure is a problem.
  Response: We may want to reward extra for restructuring.
- **Sources.** There has a been some work explicitly taking external links to be relative to quality,[**CITEHYPERLINKS**] and seems to have been bore out by a further study which found this to be a heuristic used by actual readers.[**THISHEURISTICIGUESS**].
  Response: We may give give extra weight to the value of (working) hyperlinks, and fixing hyperlinks.

[WIKIPEDIA SELF EVALUATION AND MARKUP ETC]

> [Wikipedia] cannot attain the status of a true encyclopedia without more formal content-inclusion and expert review procedures.[8]

Most visited articles in an hour - correlates with (american-centric) events [38]

Denning says it cannot attain the status of a true encyclopedia without more formal content-inclusion and expert review procedures[8] this corroborates by findings in [9]?

'robust and remarkable growth' [13][30]

Wikipedia, at the last dump, consisted 800G of compressed data [32]

Figure 4: Wikitrust in action (2011)

2.1.1 Evaluating Wikipedia articles

identify, analyse

after article mentioned in press [16]

compared by 'experts' to 'equivalent' Encyclopedia Britannica articles [9]

found metrics of article quality through factor analysis [27]

Analysis by conflict - revisions?[13]

WikiTrust. The most 'complete' of the many of the. Exists as firefox plugin (though it doesn't work any more) Culmination of various studies that try to QUOTE [2] and QUOTE CITE. It was assessed as recently as 2011 [17]

different views emerging topics gaps and inconsistencies

Goals.

Methods. Python. Levenshtein. Optimisations of.

For this study we will assume that the 'final' or 'target' version is fully trusted, or, that its quality does not need to be evaluated by us, and does not affect how we evauluate contributions. This way we can leave issues of trust regarding the article itself to one side, and concentrate on the various edits made. The reputation, or quality, of the article itself is not important for this study. Our intention is to evaluate an individual's weighted stake in an article - whether that article is of good or bad quality is something of a different issue.

However, some of the methods used to measure quality are directly applicable here, and, as mentioned previously, are copious. Of particular interest is the academic work that culminated in WikiTrust,[7][2] and other studies of significant subordinate importance.[40][6]

Wikitrust was[3] a firefox plugin, designed to highlight the words of a Wikipedia article with different gradations of yellow. The gradations relate to levels of trust, a screenshot can be seen below.

[17] Evaluating Wikitrust [18] Trust in Wikipedia (how many links)

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

3 Conclusions

Now we have reviewed the existing work, we will begin to outline our intentions. Here we discuss the assumptions that we make about our data, our intentions with regards to analysing that data, and the metrics by which we will regard each contribution.

Assumptions

- **We assume that the final, or 'target' article is of good quality.** There are many studies which concern themselves with verifying the accuracy and quality of Wikipedia articles. In this study we are specifically concerned with the quality of contribution, i.e. the quality of text within the article, relative to the domain of the article. The quality of the article itself is a moot point.
- **We assume the article is well-formed.** As much as we do not concern ourselves with article accuracy, we also assume that the Wikipedia markup is also well-formed. We may also eventually check whether links are invalid, but for the moment we assume that they are.
- **We make no distinction between humans, bots and anonymous editors.**

Weighting contribution by markup

Given the extesive research regarding which features of a wikipedia article are most important, we may define the following features to have more weight than standard text from the outset:

- Links
- Images
- Equations

We may either preprocess the text to identify the each of these different flavours of text by their Wikipedia mark-up conventions, or we may be able to more fluently work them into the main difference algorithm, raising and lowering flags during runtime.

Awarding restructuring

It has been found that, even in the most accurate articles, that the structure of Wikipedia article can be weak.[9] We should award attempts to reorganize articles. We

Awarding the dense edits

We should give extra reward to the density of the changes. For example, 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, and some weighting factor k. Something along the lines of:

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

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.

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 the figure below:

$$\begin{tikzcd}
 & v_i \arrow[dr, "ed(v_i, v_{i+1})"] & \\
v_{i-1} \arrow[ur, "ed(v_{i-1}, v_i)"] \arrow[rr, "ed(v_{i-1}, v_{i+1})"'] & & v_{i+1}
\end{tikzcd}$$

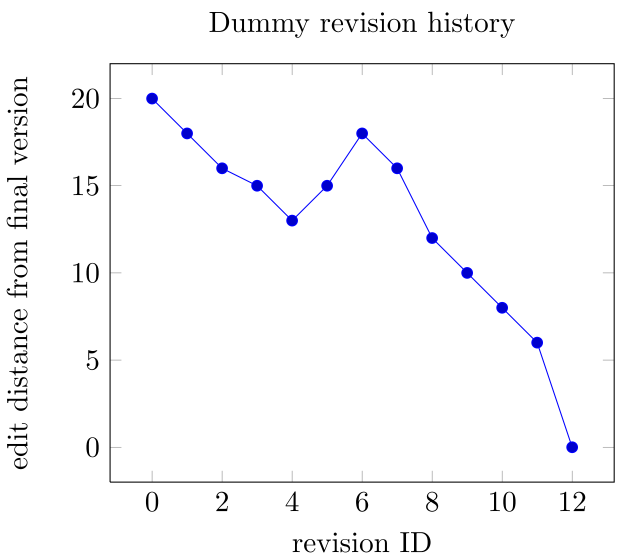
a) if $ed(v_{i-1}, v_i) < ed(v_{i-1}, v_i) + ed(v_i, v_{i+1})$, then $ed(v_{i-1}, v_i)$ has been partially undone.
b) if $ed(v_{i-1}, v_{i+1}) = 0$ $(v_{i-1} = v_{i+1})$, then $ed(v_{i-1}, v_i)$ has been completely undone.

Figure 5: Diagram showing identification of apartially or completely undone edit

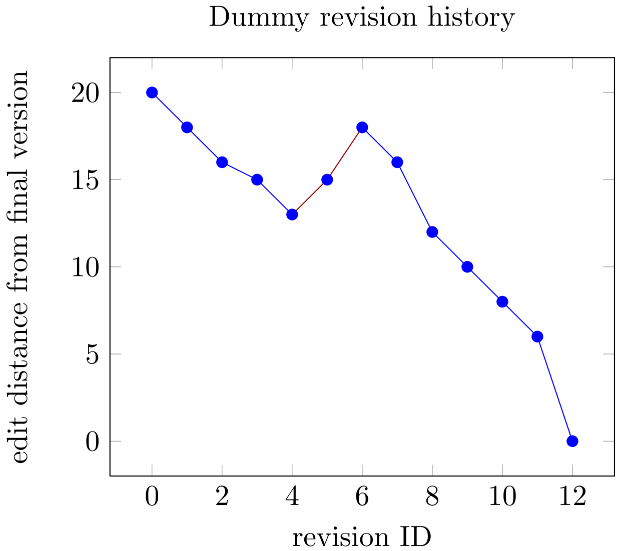
We identify immediately undone or partially undone edits according to edit distance, as layed out in Adler et al[2]. The triangle represents three consecutive revisions, and the arrows are the edit operations that transform one to another. By calculating the edit distance between v_{i-1} and v_{i+1} 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 v_i as a 'diversion'. If v_{i-1} can be transformed into v_{i+1} with less operations than the two edits that actually bridged that gap, then perhaps some of the edits in $v_{i-1} \to v_i$ were unnecessary, and undone by v_{i+1}. In this case, we may punish the edit $v_{i-1} \to v_i$ (for the diversion), reward $v_i \to v_{i+1}$, or both. Case b is an extreme version of a - the texts v_{i-1} and v_{i+1} are identical, so the changes in v_i must have been completely, and immediately, undone. These reverts are common in Wikipedia edit practice.[35]

These reverts, however, are limited in their scope. Although the system may easily be extended to cover larger spans of history, to consider all nodes in a history would require the edit-distance computation of many version pairs. I propose a different, more efficient way of characterising redundant entries in terms of longer history spans. The studies, mentioned above, that graph aim to graph a Wikipedia revision history, hint at a possible method of characterising redundant material in terms of a much larger time-frame. Again, we may utilise the fact taht we have take one article to be the ultimate destination of all previous edits.

Perhaps we may graph the entirety of a wikipedia's revision history in terms of the edit distance from this final version, thus:

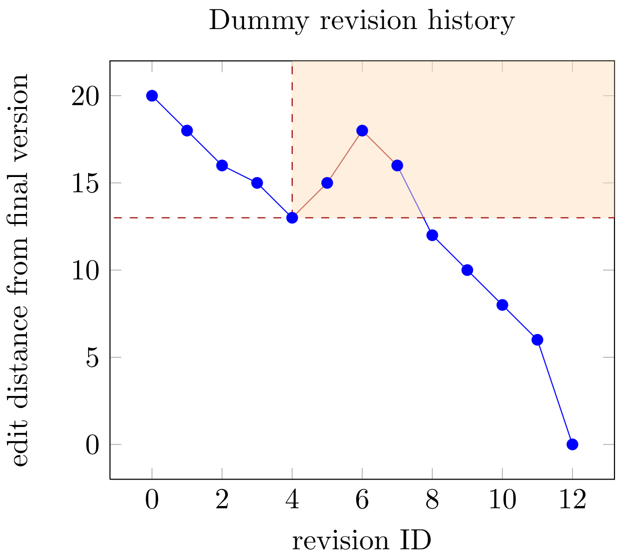


Each point represents a different version, and each line represents the the edit-distance between each version. Given this information, we may be able to disregard some information by only considering revisions that bring us closer to the final version. They appear on this graph as lines with a negative gradient; those with a positive gradient take us further away from the final version:



We may ignore the lines with a positive gradient (the two red lines). This is simple to implement. Given a version v_i, having computed it's immediate edit distance $ed(v_{i-1}, v_i)$, and it's edit distance from the final version, ed_{final_i}, we know our next computation must be $ed(v_{j-1}, ed_j)$, such that j is the smallest number that satisfies the qualities $j > i$ and $ed_{final_j} < ed_{final_{j-1}}$.

Another possible version 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.



The green area shows the area in which we look to find v_j. Here, after considering v_i, we move to v_j, such that j is the smallest number that satisfies to qualities $j > i$ and $ed_{final_j} < ed_{final_i}$. In the graph above we move from (4,13) to (8,12). It is worth stating here that always compute $ed(v_{j-1}, v_j)$, not necessarily $ed(v_i, v_j)$. We will look into the pros and cons of these different strategies further into the project.

Possible extensions

Visualisation

Given that we are distributing wealth according to a series of weight factors, it may be useful to devise a system that visualises how these weight factors affect the distribution. This would be a matter of expressing the final 'score' in terms of these variables, and producing some interactive graphs. We will look into the viability of this extension further into the project.

Data

Current intentions are to approach the data on an article-by-article basis, grabbing a particular version and tracing its history backwards. We grab the pages using HTTP requests, as described earlier (see page **??**). We could, however, download Wikipedia in its entirety. The entire site is compressed and dumped monthly, and the dumps are free to download.[32]

Further analysis

Given the over-arching nature of Wikis, it may be possible to derive other information about the articles we study. By-products of this study include various ways to visualise the histories of different articles, and comprehensive storage of various dimensions of Wikipedia artefacts and agents. We may be able compare and contrast different categories of article, or editor. It would be interesting to contrast the actions of humans, and bots,[4], and perhaps look at the nature of edits made by different groups of editors.

Further subjects

The project may well extend to subjects beyond Wikipedia. A github project, for instance may be of interest in further study. We may begin to look into software metrics, such as Cyclomatic Complexity, which measures code flow complexity,[**McCabe1976**]

It may be worth Perhaps we can examine more about what we may find out about the articles. Cite other studies Lih's 2004 study of articles immediately after they have been cited in the press[16], and Metyan's 2012 use of the site to predict box-office success [21]. Lieberman worked out their locale.[15]:

1. peaks in activity (rate of levenshtein distance... may have to be in a log graph...)
2.

PREDIFINED / NOT-PREDEFINED ideas of quality. look for when the the article levels off? And do this by DATE rather than REVISION. We may assume that pageviews are more well-distributed than revisions

summarize major contributions (which do we care about?) evaluate your current position point out any flaw in methodology/research/contradictions are there any gaps in the area which you will cover in your research? How will you integrate sources you have mentioned into your dissertation? Point out any areas for further study

[4]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 submisions in a matter of seconds.[36][23]

4 Progress report

One python scraper

One simple python levenshtein distance

One class combining both

References

References

[1] *AcaWiki*. `http://acawiki.org`. Accessed: 2014-04-31 (see p. 5).

[2] 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. 6, 8).

[3] *Alexa About Us*. `http://www.alexa.com/about`. Accessed: 2014-04-31 (see p. 4).

[4] *Alexa The top 500 sites on the web*. `http://www.alexa.com/topsites`. Accessed: 2014-04-31 (updated daily) (see p. 4).

[5] 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. 2).

[6] 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. 6).

[7] 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. 6).

[8] 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. 5).

[9] 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. 5–7).

[10] R. W. Hamming. "Error Detecting and Error Correcting Codes". In: *Bell System Technical Journal* 29.2 (1950), pp. 147–160. ISSN: 1538-7305. DOI: `10.1002/j.1538-7305.1950.tb00463.x`. URL: `http://dx.doi.org/10.1002/j.1538-7305.1950.tb00463.x` (see p. 3).

[11] James J. Hunt, Kiem-Phong Vo, and Walter F. Tichy. "Delta Algorithms: An Empirical Analysis". In: *ACM Trans. Softw. Eng. Methodol.* 7.2 (Apr. 1998), pp. 192–214. ISSN: 1049-331X. DOI: `10.1145/279310.279321`. URL: `http://doi.acm.org/10.1145/279310.279321` (see p. 4).

[12] Heikki Hyyrö. "A bit-vector algorithm for computing Levenshtein and Damerau edit distances". In: *Nordic Journal of Computing* (2003), p. 2003 (see p. 4).

[13] 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 pp. 5, 6).

[14] VI Levenshtein. “Binary Codes Capable of Correcting Deletions, Insertions and Reversals”. In: *Soviet Physics Doklady* 10 (1966), p. 707 (see p. 1).

[15] 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. 10).

[16] 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 pp. 6, 10).

[17] 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 p. 6).

[18] 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. 6).

[19] Josh MacDonald. “File system support for delta compression”. PhD thesis. Masters thesis. Department of Electrical Engineering and Computer Science, University of California at Berkeley, 2000 (see p. 4).

[20] 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. 4, 14).

[21] 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. 10).

[22] 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. 2).

[23] 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. 10).

[24] 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. 5).

[25] *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. 16).

[26] 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. 3).

[27] 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. 6).

[28] 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. 5).

[29] 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. 3).

[30] J Voss. "Measuring Wikipedia". In: *Proceedings 10th International Conference of the ...* (Jan. 2005). URL: http://lidan.net/backup/Documents/untitled\%2520folder/MeasuringWikipedia2005.pdf (see p. 5).

[31] *WikiLit*. http://wikilit.referata.com. A temporary site, relating directly to [20]. Accessed: 2014-04-31 (see p. 5).

[32] *Wikimedia enwiki dump progress on 20140502*. http://dumps.wikimedia.org/enwiki/20140502/. Accessed: 2014-04-29 (see pp. 5, 10).

[33] *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. 4).

[34] *WikiPapers*. http://wikipapers.referata.com. Accessed: 2014-04-31 (see p. 5).

[35] 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. 8).

[36] 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. 10).

[37] *Wikipedia Counter-Vandalism Unit*. https://en.wikipedia.org/wiki/Wikipedia:CVU. Accessed: 2014-04-31 (see p. 5).

[38] *Wikipedia Wikipedia Signpost/2013-02-04/Special report: Examining the popularity of Wikipedia articles: catalysts, trends, and applications*. https://en.wikipedia.org/wiki/Wikipedia:Wikipedia_Signpost/2013-02-04/Special_report. Accessed: 2014-04-31 (see p. 5).

[39] Dennis M. Wilkinson and Bernardo A. Huberman. "Assessing the Value of Coooperation in Wikipedia". In: *CoRR* abs/cs/0702140 (2007) (see pp. 4, 5).

[40] 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. 6).

A Appendix A: Code progress

A.1 Python class for scraping a Wikipedia article's revision history

The following code is a first draft of a class which incrementally traces, parses, and stores the revision history of select articles. It chooses random articles up to a maximum amount of articles unless an article (or articles) are specified. It traces the entire discoverable[5] history of the, unless a specific depth is specified by the user.

The class already yields workable data, but here is some immediate further work for this code:

- Allow the user to specify timeframe
- Allow for integration with a postgres database (at the moment the code saves the data in CSV format).

This code is an extension to an existing wikipedia API class for python (which did not provide the features we need here).[25]

```
import requests
import time
import json
import csv
import wikipedia
from bs4 import BeautifulSoup
from datetime import datetime, timedelta
from decimal import Decimal

WIKI_API_URL = 'http://en.wikipedia.org/w/api.php'
WIKI_USER_AGENT = 'wikipedia (https://github.com/goldsmith/Wikipedia/)'

class WikiRevisionScrape:
    par = {
        'format': 'json',
        'action': 'query',
        'prop': 'revisions'
        }
    head = {
        'User-Agent': WIKI_USER_AGENT
        }
    rand = True
    pagelimit = 1
    historylimit = -1
    rl = False
    rl_minwait = None
    rl_lastcall = None
    pageid = 0
    parentid = 0
    childid = 0

    #atm naively assuming headers, params, titles to be in correct format
    def __init__(self, pagelimit=1, historylimit=-1, _headers=None, _params=None,
        _titles=None):
        if(_params):
            params = _params

        if(_headers):
            self.head = _headers
```

[5]Using the Wikipedia API, articles can either be traced back to their origin (revision parent ID = 0), or to the point at which a loop is found in the revision history - this usually happens with older articles.

```
    if(_titles):
        self.params['titles'] = _titles
        self.rand = False

    self.pagelimit = pagelimit
    self.historylimit = historylimit


def scrape(self, indexfilename, contentsfilename):
    self._pace()
    index_f = open(indexfilename + ".csv", "ab") #HACK = needs to migrate to postrgres
    contents_f = open(contentsfilename + ".csv", "ab") #HACK = needs to migrate to
        postrgres
    index = csv.writer(index_f)
    contents = csv.writer(contents_f)
    index.writerow(["PAGEID","REVISION","USER","USERID","TIMSTAMP","SIZE","COMMENT"])
    contents.writerow(["PAGEID","REVISION","CONTENT"])

    for i in range(self.pagelimit):
        if 'rvprop' in self.par:
            del self.par['rvprop']
        if 'revids' in self.par:
            del self.par['revids']
        print "fetching page"
        if(self.rand):
            self.par['titles'] = wikipedia.random() #get random title
        self.childid = self._getlatest()
        r = requests.get(WIKI_API_URL, params=self.par, headers=self.head)
        self._rate()
        del self.par['titles']
        self._tracehist(index, contents)

def _getlatest(self):
    r = requests.get(WIKI_API_URL, params=self.par, headers=self.head)
    r = r.json()

    #HACK = should grab multiple pages
    for key, value in r['query']['pages'].iteritems():
        self.pageid = key
    #HACK = chould grab multiple revisions (for each pageid)
    self.parentid = self.childid =
        r['query']['pages'][self.pageid]['revisions'][0]['revid']
    return self.childid

def _tracehist(self, index, contents):
    ##We store revisions we've visited
    ##loops can occur in revision histories
    visited = []
    i = self.historylimit
    j = 0

    self.par['rvprop'] =
        'userid|user|ids|flags|tags|size|comment|contentmodel|timestamp|content'

    while (self.parentid not in visited) and i is not 0 and self.parentid is not 0:
        self.par['revids'] = self.parentid

        self._pace()

        r = requests.get(WIKI_API_URL, params=self.par, headers=self.head)
```

```
            r = r.json()

            self._rate()

            visited.append(self.childid)

            #print r

            self.childid = r['query']['pages'][self.pageid]['revisions'][0]['revid']
            self.parentid = r['query']['pages'][self.pageid]['revisions'][0]['parentid']
            user = r['query']['pages'][self.pageid]['revisions'][0]['user']
            userid = r['query']['pages'][self.pageid]['revisions'][0]['userid']
            size = r['query']['pages'][self.pageid]['revisions'][0]['size']
            timestamp = r['query']['pages'][self.pageid]['revisions'][0]['timestamp']
            comment = "" #comments sometimes don't return from old revisions...
            try:
                comment = r['query']['pages'][self.pageid]['revisions'][0]['comment']
            except:
                comment = ""
            content = r['query']['pages'][self.pageid]['revisions'][0]['*']

            index.writerow([self.pageid, self.childid, user.encode("UTF-8"), userid,
                timestamp, size, comment.encode("UTF-8")])
            contents.writerow([self.pageid, self.childid, content.encode("UTF-8")])

            if(self.historylimit > 0):
                print self.pageid, "fetch", j+1, "of", self.historylimit, ", revid",
                    self.childid, "timestamp", str(timestamp)
                i = i - 1
            else:
                print self.pageid, "fetch", j+1, ", revid", self.childid, "timestamp",
                    str(timestamp)
            j = j + 1
        print "limit reached"

    def _pace(self):
        if self.rl and self.rl_last_call and self.rl_lastcall + self.rl_minwait >
            datetime.now():
            wait_time = (self.rl_lastcall + self.rl_minwait) - datetime.now()
            time.sleep(int(wait_time.total_seconds()))

    def _rate(self):
        if self.rl:
            self.rl_lastcall = datetime.now()
```

A.2 Example output

```
$ python
>>> import WikiRevisionScrape

>>> singlescraper = WikiRevisionScrape.WikiRevisionScrape()

>>> singlescraper.scrape("filename1","filename2")
fetching page
searching for
{u'action': u'query', u'list': u'random', u'rnlimit': 1,
     u'rnnamespace': 0, u'format': u'json'}
```

```
25455543 fetch 1 , revid 553292956 timestamp 2013-05-03T03:01:26Z
25455543 fetch 2 , revid 550043052 timestamp 2013-04-12T18:59:57Z
25455543 fetch 3 , revid 503496279 timestamp 2012-07-21T21:52:51Z
.
. [skipping some output]
.
25455543 fetch 23 , revid 331902859 timestamp 2009-12-15T23:25:23Z
25455543 fetch 24 , revid 331902368 timestamp 2009-12-15T23:22:50Z
25455543 fetch 25 , revid 331902181 timestamp 2009-12-15T23:21:47Z
limit reached

>>> multiscraper = WikiRevisionScrape.WikiRevisionScrape(pagelimit=1000)

>>> multiscraper.scrape("multifilename1","multifilename2")
fetching page
searching for
{u'action': u'query', u'list': u'random', u'rnlimit': 1,
     u'rnnamespace': 0, u'format': u'json'}
7096591 fetch 1 , revid 472732138 timestamp 2012-01-23T03:00:01Z
7096591 fetch 2 , revid 416290467 timestamp 2011-02-28T00:06:47Z
.
. [skipping some output]
.
7096591 fetch 8 , revid 89546539 timestamp 2006-11-22T23:31:09Z
7096591 fetch 9 , revid 77039186 timestamp 2006-09-21T20:00:55Z
limit reached
fetching page
searching for
{u'action': u'query', u'list': u'random', u'rnlimit': 1,
      u'rnnamespace': 0, u'format': u'json'}
24830105 fetch 1 , revid 547881527 timestamp 2013-03-30T21:34:39Z
24830105 fetch 2 , revid 500160388 timestamp 2012-07-01T09:55:31Z
.
. [skipping some output]
.
[etc.]
```

B Appendix B: Python Levenshtein distance implementation

This Python class gives a basic implementation of Levenshtein distance. To compare strings x and y both its time and space complexity is $\Theta log(|x| \bullet |y|)$.

The class is instantiated with the twwo strings, or files, it is to compare. Methods can then be accessed in order to examine the distance between the provided strings. The class provides command-line visualisations of the data — it can print out its table of computations, as well as instructions on how to transform one into the other. Please see the exmple output below. It does not currently give any information about optimal alignment, although information about this alignment is found in the table.

B.1 Code

```
import sys

class LevDistBasic:
    e = [] #edit operation array
    t = [] #grid array
    x = "" #string1
    y = "" #string2
    m = 0 #length string1
    n = 0 #length string2
    dist = 0 #Levenshtein distance
    ed = [] #the edit operation, calculated in _calculate()
    isFile = False

    def __init__(self, _x, _y, isFile=False):
        self.x = self._variablehandle(_x)
        self.y = self._variablehandle(_y)
        self.m = len(self.x)
        self.n = len(self.y)
        self.t = [[0]*(self.n+1) for _ in xrange(self.m+1)]
        self.e = [[" "]*(self.n+1) for _ in xrange(self.m+1)]
        self.dist = self._calculate()

    def __str__(self):
        return str(self.distance())

    def distance(self):
        return self.dist

    def strings(self):
        return self.x, self.y

    def table(self):
        return self.t

    def operation(self):
        return self.ed

    ##ADD WARNING for long strings / deal with them
    def showtable(self):
        result = ""
        for ch in self.y:
            result = result + ch + " "
        print "      ", result
        for r in range(len(self.t)):
```

```
        s = ' '
        if r:
            s = self.x[r-1]
        print s, ' ', self.t[r]

def showop(self):
    for i, op in enumerate(self.ed):
        l = str(i) + ": "
        if op[0] == 'I':
            l += "insert " + op[-1]
        elif op[0] == 'K':
            l += "keep " + op[-1]
        elif op[0] == 'D':
            l += "delete " + op[-1]
        elif op[0] == 'S':
            l += "swap " + op[-1][0] + " for " + op[-1][-1]
        else:
            return "FAIL: incorrect operation"
        print l

def _ed(self):
    i, j = len(self.e)-1, len(self.e[0])-1
    self._ed_recursive(i,j)

def _ed_recursive(self,i,j):
    if self.e[i][j] == ' ':
        if i == 0 and j > 0:
            self.ed.append(('D', self.y[0]))
        if j == 0 and i > 0:
            self.ed.append(('D', self.x[0]))
        return
    if self.e[i][j] == 'K':
        self._ed_recursive(i-1, j-1)
        self.ed.append((self.e[i][j], self.x[i-1]))
    elif self.e[i][j] == 'S':
        self._ed_recursive(i-1, j-1)
        self.ed.append((self.e[i][j], (self.x[i-1] + ',' + self.y[j-1])))
    elif self.e[i][j] == 'D':
        self._ed_recursive(i-1,j)
        self.ed.append((self.e[i][j], self.x[i-1]))
    else:
        self._ed_recursive(i,j-1)
        self.ed.append((self.e[i][j], self.y[j-1]))

def _variablehandle(self,v):
    if not isinstance(v, str):
        try:
            return v.read()
        except:
            try:
                return str(v)
            except:
                print "Argument cannot be of type" + type(v)
                raise
            pass
    return v

def _calculate(self):
    for i in xrange(self.m+1):
        self.t[i][0] = i
    for j in xrange(self.n+1):
```

```
            self.t[0][j] = j
        j = 1
        while j < self.n+1:
            i = 1
            while i < self.m+1:
                c = (self.x[i-1] != self.y[j-1])
                dl = self.t[i-1][j] + 1
                ins = self.t[i][j-1] + 1
                sbs = self.t[i-1][j-1] + c
                self.t[i][j] = min(ins, dl, sbs)
                if ins < dl and ins < sbs:
                    self.e[i][j] = 'I'
                elif dl <= sbs:
                    self.e[i][j] = 'D'
                else:
                    if(self.x[i-1] != self.y[j-1]):
                        self.e[i][j] = 'S'
                    else:
                        self.e[i][j] = 'K'
                i += 1
            j += 1
        self._ed()
        return self.t[self.m][self.n]
```

B.2 Example output

```
$ python
>>> import LevDistBasic
>>> test = LevDistBasic.LevDistBasic("bank","book")
>>> test.showtable()

        b  o  o  k
    [0, 1, 2, 3, 4]
b   [1, 0, 1, 2, 3]
a   [2, 1, 1, 2, 3]
n   [3, 2, 2, 2, 3]
k   [4, 3, 3, 3, 2]

>>> t = test.table()
>>> print t

[[0, 1, 2, 3, 4], [1, 0, 1, 2, 3], [2, 1, 1, 2, 3], [3, 2, 2, 2, 3],
[4, 3, 3, 3, 2]]

>>> s = test.strings()
>>> print s

('bank', 'book')

>>> test.showop()

0: keep b
1: swap a for o
2: swap n for o
```

```
3: keep k

>>> ed = test.operation()
>>> print ed

[('K', 'b'), ('S', 'a,o'), ('S', 'n,o'), ('K', 'k')]

>>> print test

2
```