Computing Wikipedia Edit-Networks

Ulrik Brandes
University of Konstanz, Germany
Ulrik.Brandes@uni-konstanz.de

Patrick Kenis
TiasNimbas Business School & Tilburg University, The Netherlands
p.kenis@tiasnimbas.edu

Jürgen Lerner[*]
University of Konstanz, Germany
lerner@inf.uni-konstanz.de

Denise van Raaij
Tiburg University, The Netherlands
D.P.A.M.Korssen-vanRaaij@uvt.nl

ABSTRACT

This technical paper reviews the definition of Wikipedia edit-networks proposed in [1] and presents an algorithm to compute them.

1. THE EDIT-NETWORK

In a nutshell, the edit-network associated to a Wikipedia page p has as nodes the authors of p and encodes how authors contributed to p and how authors interacted with each other while editing p. This information is computed from the complete history of p, i. e., from the sequence of its revisions, by determining which part of the text has been added, has been deleted, or remained unchanged when going from one version of the page to the next. Since an exhaustive description of the text-processing algorithm is rather technical, we first describe precisely *what* information is encoded in the edit-network. We clarify in Sect. 2.1 how we handle reverts, duplicated text, and text that has been moved to a different location on the page. An algorithm to compute edit-networks is detailed in Sect. 2.

1.1 Network Model

The *edit-network* associated to a Wikipedia page p is a tuple $G = (V, E, \mathcal{A})$, whose components are defined as follows.

1. The *nodes* V of the graph (V, E) correspond to the authors that have done at least one revision on p.
2. The *directed edges* $E \subseteq V \times V$ of the graph (V, E) encode the edit interaction among authors. A particular pair of authors $(u, v) \in V \times V$ is in E, if u performed one of the following three actions with respect to v.
   (a) u *deletes* text that has been written by v;
   (b) u *undeletes* text that has been deleted by v (and written by a potentially different author w);
   (c) u *restores* text that has been written by v (and deleted by a potentially different author w).

   Since authors may as well revise text written by themselves, *loops*, i. e., edges connecting an author with herself, are allowed.
3. $\mathcal{A}$ is a set of *weighted attributes* on nodes and edges, explained in Sect. 1.2.

1.2 Attributes on Nodes and Edges

The weighted attributes on nodes and edges encode how much text users add, delete, or restore. Furthermore, in case of deletion we keep track of who has previously written the text and in case of restoration we keep track of both, the original author and the deleter of the restored text. By combining these attributes, we can get deep insight into the various roles that users play when editing, as well as into relations between users, see Sect. 1.2.2. The *amount of text* added, deleted, or restored is measured by the number of words.

We will also keep track of the timepoint when edit-actions occur by indexing attributes with the revision number. In the following we assume that the history of a given page is a sequence of revisions $R = (r_1, \dots, r_N)$, ordered by increasing timestamps $1, \dots, N$.

1.2.1 Basic Attributes

Basic attributes are those that have to be computed by the network construction algorithm; others can be derived from these, see Sect. 1.2.2.

Attributes on edges. For each timepoint $i \in \{1, \dots, N\}$ and each pair of authors $(u, v) \in V \times V$,

- $\texttt{delete}_i(u, v)$ denotes the number of words deleted by u in revision r_i and written by v at earlier timepoints $j\,(j < i)$;
- $\texttt{undelete}_i(u, v)$ denotes the number of words restored by u in revision r_i, deleted by v at timepoints $j\,(j < i)$, and written by a potentially different author w at timepoints $\ell\,(\ell < j < i)$;
- $\texttt{restore}_i(u, v)$ denotes the number of words restored by u in revision r_i, written by v at timepoints $j\,(j < i)$, and deleted by a potentially different author w at timepoints $\ell\,(j < \ell < i)$.

Note that $\texttt{delete}_i(u, v)$, $\texttt{undelete}_i(u, v)$, and $\texttt{restore}_i(u, v)$, are equal to zero, if u is not the author of revision r_i.

Attributes on nodes. For each timepoint $i \in \{1, \dots, N\}$ and each author $u \in V$,

[*]corresponding author

- $\texttt{add}_i(u)$ denotes the number of words that are added by u at time i;
- $\texttt{authorship}_i(u)$ denotes the number of words in revision r_i that have been authored by u, i. e., all words that have been added to the text by u in a revision $j \leq i$ and that are still there in r_i.

If u is not the author of r_i, then $\texttt{add}_i(u)$ equals zero. Note however, that even in this case $\texttt{authorship}_i(u)$ might be bigger than zero. It always holds that $\texttt{add}_i(u) \leq \texttt{authorship}_i(u)$ since, at time i, u is the author of at least those words that she added in r_i and it holds that $\texttt{authorship}_i(u) \leq \sum_{j=1}^{i} \texttt{add}_j(u)$ since, at time i, u can only be the author of those words that she added before or at the i'th revision.

1.2.2 Derived Attributes

Starting from the basic attributes we can define a number of other attributes that characterize pairs of authors (dyadic variables, defined on edges) or single authors (monadic variables, defined on nodes). In this section we define these attributes and describe how they are interpreted.

Attributes on edges. Summing values over all timepoints yields three weight functions for edges $(u, v) \in E$, that are given by

$$\texttt{delete}(u,v) \quad = \quad \sum_{i=1}^{N} \texttt{delete}_i(u,v) \ \ ,$$

and $\texttt{undelete}(u,v)$ and $\texttt{restore}(u,v)$ by similar formulas.

Large values of the weights $\texttt{delete}(u,v)$ and $\texttt{undelete}(u,v)$ imply a *negative* relationship from u to v. Indeed, if u deletes a lot of text written by v, then u apparently disagrees with v's contributions to the article. Similarly, if u undeletes a lot of text that has been previously deleted by v, then u disagrees with v removing this text from the article. On the other hand, large values of the weight $\texttt{restore}(u,v)$ imply a *positive* relationship from u to v, since u defends v's contributions against deletion.

The sum over the two negative relations, denoted by

$$\texttt{revise}(u,v) = \texttt{delete}(u,v) + \texttt{undelete}(u,v) \ \ ,$$

encodes how much u undoes v's edits. It is interpreted as a measure of how much u *disagrees* with v. Similarly, $\texttt{restore}(u,v)$ is interpreted as a measure of how much u *agrees* with v.

Attributes on nodes. Summing the `delete` and `restore` weights of all edges starting from one author u give measures of how many words u deletes, respectively restores. Note that it is unnecessary to count the number of undeleted words, since this count is equal to the number of restored words.

$$\begin{aligned}
\texttt{delete}_i(u) &= \sum_{v \in V} \texttt{delete}_i(u,v) \ \ ; \\
\texttt{restore}_i(u) &= \sum_{v \in V} \texttt{restore}_i(u,v) = \sum_{v \in V} \texttt{undelete}_i(u,v)
\end{aligned}$$

Summing the attributes `add`, `delete`, and `restore` over all timepoints $i = 1, \ldots, N$ defines attributes that summarize the editing-work of the respective author, given by

$$\texttt{add}(u) \quad = \quad \sum_{i=1}^{N} \texttt{add}_i(u)$$

and $\texttt{delete}(u)$ and $\texttt{restore}(u)$ by similar formulas. The attributes $\texttt{add}(u)$, $\texttt{delete}(u)$, and $\texttt{restore}(u)$ characterize u's *role* as being a provider of new content, someone who removes content, or someone who defends content from being deleted.

At the last timepoint N it holds that $\texttt{authorship}_N(u) \leq \texttt{add}(u)$, since u can only be author of those words that she added. (Note that we adopt the convention that u does not become the author of a word that she restores. Rather, the author who has originally written that word before deletion gets her `authorship` function increased by one.) Normally $\texttt{authorship}_N(u)$ will be smaller than $\texttt{add}(u)$, since many words written by u might be deleted afterwards.

A measure of *involvement* of authors in the Wikipedia article they edit is given by the sum

$$\texttt{activity}(u) = \texttt{add}(u) + \texttt{delete}(u) + \texttt{restore}(u) \ \ ,$$

called the *edit-activity* of author u. It is the number of words that u touched by adding, deleting, or restoring them.

A further characterization of *how* an author u contributes to a page is given by the difference

$$\texttt{netadded}(u) = \texttt{add}(u) + \texttt{restore}(u) - \texttt{delete}(u) \ \ ,$$

called the *net-amount of added words.* It is the number of words by which u increased the length of the text. If `netadded(u)` is positive, then u's intention was apparently either to increase the text by adding new words or prevent it from being shortened by restoring deleted text. If `netadded(u)` is negative, then u's intention was apparently to decrease the length of the text by deleting parts of it. The absolute value of `netadded` is always bounded by `activity`. Thus, for any author with non-zero `activity`, the ratio

$$\texttt{netaddedratio}(u) = \texttt{netadded}(u)/\texttt{activity}(u)$$

lies between minus one and plus one. If $\texttt{netaddedratio}(u) = -1$, then u dedicates all her activity to deletion of text and if $\texttt{netaddedratio}(u) = 1$, then u dedicates all her activity to either adding or restoring text.

We call the value

$$\texttt{revisor}(u) = \sum_{v \in V} \texttt{revise}(u,v)$$

(i. e., u's weighted out-degree with respect to the `revise` relation) u's *degree as a revisor.* It is the number of words that u deletes after they have been added, or restores after they have been deleted, i. e., it is a measure that characterizes the *undo-activity* of v. Similarly,

$$\texttt{revised}(u) = \sum_{v \in V} \texttt{revise}(v,u)$$

(i. e., u's weighted in-degree with respect to the `revise` relation) is called u's *degree as being revised.* It is a measure of how much u's edits are undone later. It holds that both, $\texttt{revisor}(u)$ and $\texttt{revised}(u)$, are bounded from above by $\texttt{activity}(u)$. Actors with $\texttt{revisor} \approx \texttt{activity}$ show a *reactive* behavior, since they dedicate most of their activity to undo changes made by others. In contrast, actors with revised $\approx$ activity are those that do not succeed in making any of their edits permanent, as these are mostly undone afterwards.

2. COMPUTING THE EDIT-NETWORK

In this section we describe in detail how the revision network defined in Sect. 1 is determined from the history of a Wikipedia page, i. e., from the sequence of its revisions.

2.1 Text-Processing Conventions

In this section we describe the conventions that we adopt when processing the text, especially how to treat duplicated text, text that is cut and pasted to a different location, and edits that are reverts.

We consider the granularity of authorship on the word level, i. e., each word has exactly one author and different words may have different authors.

The main point to clarify is whether the ordering of words is taken into account and how to treat duplicated text. Considering the ordering of words has an obvious disadvantage in the context of Wikipedia pages: if an author restructures the page by cutting and pasting large parts of the text to different places, then we would count this as massive deletion of text and addition of newly created text. On the other hand, if we modeled the whole text as an unordered set of words, it would be impossible to determine authorship of duplicated words. (Accounting all instances of a word to the first author who has written it would also be a misinterpretation.) To overcome this dilemma, we propose to make use of the fact that words are assembled to sentences and we make here the assumption that one sentence represents one statement, gives one fact, or makes one claim. More precisely, we will model the whole text as an unordered *set* of sentences, which in turn are modeled as ordered lists of words. In particular, moving a complete sentence to another position, duplicating a complete sentence, or deleting a duplicated sentence is not considered as a change. Note however, that two words within the same sentence might have different authors. For instance, if an author changes a sentence *partially* by adding some words to it, she becomes only the author of the newly added words.

Punctuation and capitalization is used only to determine the boundaries of sentences. Once the text has been split into sentences, punctuation and capitalization is ignored.

The last point we have to consider is that many edits on Wikipedia pages are so-called *reverts*, i. e., edits that set back the page to an earlier version. For instance, if a user u deletes the whole content of a page in revision i and another user v restores it in revision $i+1$ to the version $i-1$, it would not be reasonable to credit v as the author of the whole text (actually, she has not written it and might be completely ignorant of the topic). Rather, we set back the authorship of all words as it has been assigned in revision $i-1$ and just keep track of the fact that v undeletes a lot of text deleted by u.

Table 1 gives an example of four revisions and the resulting authorship of words as determined by the conventions listed above. In this example Greek letters stand for words and periods delimit sentences. Note that the third revision is interpreted in the way that Charlie interchanged the first and second sentence, deleted γ in sentence $\alpha\ \gamma\ \delta$, and changed the word α in sentence $\alpha\ \beta$ to γ. The interchange of the two sentences is established by the fact that sentence $\alpha\ \gamma\ \delta$ and sentence $\alpha\ \delta$ have a common subsequence of length two and are, therefore, the most similar pair of sentences. After the third revision, Charlie is the author of γ, Alice is the author of β, and Bob is the author of α and δ. Furthermore, it is accounted that Charlie deleted one word of Alice (namely the α from $\alpha\ \beta$), deleted one word of Bob (namely the γ from $\alpha\ \gamma\ \delta$), and added the γ in sentence $\gamma\beta$. The forth revision is a revert that makes Charlie's changes undone. Therefore, Alice deleted Charlie's γ, restored her own α, and restored Bob's γ, setting $\texttt{undelete}(\texttt{Alice}, \texttt{Charlie}) = 2$.

Table 1: Example of four revisions on a page. Greek letters stand for words.

author	text	authorship of words
Alice	$\alpha\ \beta.$	$\texttt{A}(\alpha\ \beta)$
Bob	$\alpha\ \beta.\ \ \alpha\ \gamma\ \delta.$	$\texttt{A}(\alpha\ \beta)$, $\texttt{B}(\alpha\ \gamma\ \delta)$
Charlie	$\alpha\ \delta.\ \ \gamma\ \beta.$	$\texttt{B}(\alpha\ \delta)$, $\texttt{C}(\gamma)$, $\texttt{A}(\beta)$
Alice	$\alpha\ \beta.\ \ \alpha\ \gamma\ \delta.$	$\texttt{A}(\alpha\ \beta)$, $\texttt{B}(\alpha\ \gamma\ \delta)$

2.2 Input, Datastructures, and Output

A *revision*, of a Wikipedia-page, is a tuple of the form

$$r = (\texttt{time}, \texttt{author}, \texttt{text})\ ,$$

where time is the exact timestamp of the revision (given by the second), author is a real user-name if the contributor of the revision has been logged in and it is an IP-address if the revision has been done anonymously, and text is the complete text (i. e., not just the updated parts of it) of the page, after this revision. Our algorithm gets as input a sequence $R = (r_1, \ldots, r_N)$ of revisions $r_i = (\texttt{time}_i, \texttt{author}_i, \texttt{text}_i)$ on the same page, ordered by increasing timestamps. The sequence of all revisions of a Wikipedia page (until the current timepoint) is called its *history.*

When processing the history, each revision r_i, $i = 1, \ldots, N$ will successively by augmented by an unordered set $S_i = \{s_{i1}, \ldots, s_{i\ell_i}\}$ of sentences s_{ij}. Each sentence

$$s_{ij} = (w_{ij1}, \ldots, w_{ijk_{ij}})$$

is an ordered list of pointers to words w_{ijh}. Each word w is modeled as a triple of the form

$$w = (\texttt{charseq}(w), \texttt{author}(w), \texttt{deleter}(w))\ ,$$

where $\texttt{charseq}(w)$ is the sequence of characters that make up the word (identifying the word when comparing parts of text), $\texttt{author}(w)$ is a pointer to the author who has written it, and $\texttt{deleter}(w)$ is a pointer to the author who has deleted it. If the word has not yet been deleted, or if it has been undeleted afterwards, then the deleter variable might be arbitrarily set.

A new word $w = (\texttt{charseq}(w), \texttt{author}(w), \texttt{deleter}(w))$ is only instantiated if it is newly added. In particular, if a complete sentence $s_{ij} = (w_{ij1}, \ldots, w_{ijk_{ij}})$ is copied from revision i to revision $i+1$, then no new word objects are instantiated. Rather, the set of sentences S_{i+1} contains a sentence that is the identical list of pointers as s_{ij}.

Additionally, while processing the sequence of revisions, we successively build up the edit-network $G = (V, E, \mathcal{A})$. When processing the revision $r_i = (\texttt{time}_i, \texttt{author}_i, \texttt{text}_i, S_i)$, we add $u = \texttt{author}_i$ to V (if not already in) and compute the attributes $\texttt{add}_i(u)$, $\texttt{delete}_i(u, v)$, $\texttt{undelete}_i(u, v)$, and $\texttt{restore}_i(u, v)$, for all authors $v \in V$. Note that all other attributes of the edit-network can be computed from those basic attributes, or from the authorship associated to words, respectively. We assume that the counts $\texttt{add}_i(u)$, $\texttt{delete}_i(u, v)$, $\texttt{undelete}_i(u, v)$, and $\texttt{restore}_i(u, v)$ are initialized with zero and are incremented while processing the text.

2.3 Algorithm

2.3.1 Processing the First Revision

Let $r_1 = (\mathtt{time}_1, \mathtt{author}_1, \mathtt{text}_1)$ be the first revision of the Wikipedia page. Put $u = \mathtt{author}_1$ into the set of authors V. Split $\mathtt{text}_1$ into sentences, remove duplicated sentences, and split all sentences into words (i. e., sequences of characters delimited by whitespaces). For each of these character sequences create a new instance w of a word with $\mathtt{author}(w)$ pointing to u and increment $\mathtt{add}_1(u)$ by one. Create for each sentence a list of pointers to its words and put it into the set of sentences S_1.

2.3.2 Processing Subsequent Revisions

Assume that revisions $r_1, \ldots, r_i$, $i \geq 1$, are already processed and let $r_{i+1} = (\mathtt{time}_{i+1}, \mathtt{author}_{i+1}, \mathtt{text}_{i+1})$ be the next revision. For the remainder of this Sect. 2, we write u for $\mathtt{author}_{i+1}$. Put u into V (if not already in).

Determine whether r_{i+1} is a revert. Compare $\mathtt{text}_{i+1}$ with $\mathtt{text}_j$, $j = i, \ldots, 1$ (in that order) for equality. If $\mathtt{text}_{i+1}$ equals $\mathtt{text}_j$ for some $j = i, \ldots, 1$, go to Paragraph 2.3.2.2 if not, continue with Paragraph 2.3.2.1. (Efficient equality testing of texts can, for instance, be done by using hashcodes.)

2.3.2.1 Handling revisions that are not reverts.

If the revision r_{i+1} is not a revert, we compare its text to the set of sentences S_i of revision r_i in order to determine which words have been copied, added, or deleted.

Initialize a temporary set of sentences. Let S'_{i+1} be the set of sentences determined from $\mathtt{text}_{i+1}$ as in Sect. 2.3.1. So far, the authorship of words in S'_{i+1} is undetermined. Create an empty set of sentences S_{i+1} that will be filled in the following steps. At the end of Paragraph 2.3.2.1 the temporary set S'_{i+1} is discarded.

Handle sentences that have been copied from r_i. For each sentence $s \in S_i$ that is also in S'_{i+1}, create a list of pointers to words identical to s and put it into S_{i+1} (thereby these words have the same authors as in revision i). Mark sentence s as processed both in S_i and in S'_{i+1}. No words are added and no edges induced by copied sentences.

Process successively the most similar pairs of sentences. While there are still unprocessed sentences in S_i and in S'_{i+1}, let (s, s') be the pair of unprocessed sentences $s \in S_i$ and $s' \in S'_{i+1}$ with the *longest common subsequence* [2] among all those pairs. We consider s' as a slightly changed version of s and have to determine which words have been copied, deleted, or added when going from s to s'. Mark s and s' as processed and compute a shortest *edit-script* from s to s' [3].

An edit-script is a sequence $((w_1, a_1), \ldots, (w_k, a_k))$ of pairs (w_j, a_j), where the w_j are words and the a_j are *edit-actions* that are either NONE, DELETE, or ADD. If a_j is NONE, then word w_j appears in both sentences, if a_j is DELETE, then appears in s but not in s', and if a_j is ADD, then w_j appears not in s but in s'. The order of the edit-script matters. For instance, if only one word w has been moved from the beginning of a sentence to the end, then the first pair in the corresponding edit-script is (w, DELETE) and the last one is (w, ADD). Together, all words that are labeled by NONE and DELETE make up the sentence s and all words that are labeled by NONE and ADD make up the sentence s'.

Create an empty sentence s^* (i. e., an empty list of pointers to words) and go over the edit-script from the first to the last index. If the next pair in the edit script is (w, NONE), then create a pointer to w and add it to the end of the sentence s^*. (Thereby the author of w does not change.) If the next pair in the edit script is (w, DELETE), then let v be the author of w. Increment $\mathtt{delete}_{i+1}(u, v)$ by one and let the $\mathtt{deleter}$-variable of w point to u. (No pointer is added to the sentence s^*.) If the next pair in the edit script is (w, ADD), then increment $\mathtt{add}_{i+1}(u)$ by one, create a new instance of a word from the sequence of characters w, set its $\mathtt{author}$-pointer to u, and add a pointer to that word at the end of the sentence s^*. If all pairs of the edit-script have been processed, put s^* into S_{i+1}.

Process deleted sentences. These are all sentences in S_i that are still unprocessed. For each such sentence $s = (w_1, \ldots, w_k)$ go over all words w_h, $h = 1, \ldots, k$, increment $\mathtt{delete}_{i+1}(u, \mathtt{author}(w_h))$ by one and set the $\mathtt{deleter}$-pointer of w_h to u.

Process added sentences. These are all sentences in S'_{i+1} that are still unprocessed. For each such sentence $s = (w_1, \ldots, w_k)$ create a list of pointers s^*, go over all words w_h, $h = 1, \ldots, k$, increment $\mathtt{add}_{i+1}(u)$ by one, create a new instance of a word from the sequence of characters w_h, set its $\mathtt{author}$-pointer to u, and add a pointer to that word at the end of the sentence s^*. When all words of s have been processed, add s^* to S_{i+1}.

2.3.2.2 Handling reverts.

Let $\mathtt{text}_{i+1}$ equal $\mathtt{text}_j$ and let $j < i + 1$ be the largest index with this property. Create a set of sentences S_{i+1} by copying all lists of pointers of S_j. Thereby all authors in the $i+1$'th revision are exactly as in the j'th revision. No words are added by a revert, i. e., $\mathtt{add}_{i+1}(u)$ is not increased. What remains to do is to update the attributes $\mathtt{delete}_{i+1}(u, \cdot)$, $\mathtt{undelete}_{i+1}(u, \cdot)$, and $\mathtt{restore}_{i+1}(u, \cdot)$.

For doing this, we compare the set S_{i+1} with the set S_i in almost the same way as we compared S'_{i+1} to S_i in Paragraph 2.3.2.1. The steps that are exactly identical to the corresponding steps in Paragraph 2.3.2.1 are the processing of copied and deleted words.

The only steps that are slightly changed are those that process added words. The difference lies in the fact that, in case of a revert, these words are not added (as in Paragraph 2.3.2.1) but they are restored. For any restored word w let v be the author of w and let v' be the user who deleted w at some timepoint between j and $i + 1$ (this author is pointed at by the $\mathtt{deleter}$ variable of w). Increase the counts $\mathtt{restore}_{i+1}(u, v)$ and $\mathtt{undelete}_{i+1}(u, v')$ by one.

3. REFERENCES

[1] U. Brandes, P. Kenis, J. Lerner, and D. van Raaij. Network analysis of collaboration structure in Wikipedia. In *Proc. 18th Intl. World Wide Web Conf. (WWW2009)*, 2009.

[2] D. S. Hirschberg. Algorithms for the longest common subsequence problem. *Journal of the ACM*, 24(4):664–675, 1977.

[3] E. W. Myers. An $\mathcal{O}(nd)$ difference algorithm and its variations. *Algorithmica*, 1(1):251–266, 1986.