Stable Word-Clouds for Visualising Text-Changes Over Time

Stable Word-clouds for Visualising Text-changes over Time

Elisa Herold$^{1[0000-0002-7587-3960]}$, Marcus Pöckelmann$^{1[0000-0001-8718-1198]}$, Christian Berg$^{1[0000-0003-0192-5569]}$, Jörg Ritter$^{1[0000-0003-3080-7801]}$, and Mark M Hall$^{1[0000-0003-0081-4277]}$

Martin-Luther-University Halle-Wittenberg, Halle, Germany
elisa.herold@student.uni-halle.de,
{marcus.poeckelmann,christian.berg,joerg.ritter,mark.hall}
@informatik.uni-halle.de

Abstract. Word-clouds are a useful tool for providing overviews over texts, visualising relevant words. Multiple word-clouds can also be used to visualise changes over time in a text. This requires that the words in the individual word-clouds have stable positions, as otherwise it is very difficult so see what changed between two consecutive word-clouds. Existing approaches have used coordinated positioning algorithms, which do not allow for their use in an online, dynamic context. In this paper we present a fast word-cloud algorithm that uses word orthogonality to determine which words can share the same space in the word-clouds combined with a simple, but fast spiral-based layout algorithm. The evaluation shows that the algorithm achieves its goal of creating series of word-clouds fast enough to enable use in an online, dynamic context.

Keywords: text analysis · visualisation · visual analysis · word-clouds · optimisation.

1 Introduction

Word-clouds, such as Wordle [7], have long been used to generate overviews over texts, visualising the most frequently occurring or most relevant words, generally using font size to visualise the words' frequency or importance and optionally adding colour or transparency to visualise further attributes [17].

While word-clouds work well as overview visualisations for individual texts or larger text corpora [8,22], one of the main limitations of basic word-clouds is that they only show a static view of the text. In contrast to this, for text analysis it is often desired to be able to visualise part of the text and then move the visualisation window over the text to see how the text changes [18]. While it is possible to generate individual word-clouds for each time-slice the user moves the visualisation window to, the word locations in the word-clouds will change each time, making it difficult to track changes. While algorithms such as Word Storm [3] can generate multiple word-clouds that have relatively stable word locations, the process of creating these coordinated word-clouds is time consuming.

The LERA[1] tool provides an environment for this kind of inter-textual analysis. The user can select a range within the text and then gets a word-cloud for that text range. They can then move the range through the text and observe the changes, which means that the word-clouds have to be generated very quickly. While the text itself is static and thus would be amenable to using pre-calculation of the word-clouds using the existing, slower algorithms, the user is provided with a range of parameters to tune what exactly they see in the word-cloud. The range of parameters make pre-calculation unfeasible, it is thus necessary to have a fast word-cloud algorithm that generates stable word-clouds.

In this paper we present a novel algorithm for fast generation of coordinated word-clouds in which all words have a completely fixed location. The remainder of the paper is structured as follows: Section 2 discusses previous approaches to creating comparable word-clouds, Section 3 provides an overview over the data we are working with, Section 4 presents our algorithm for stable word-clouds, which is evaluated in Section 5 and in Section 6 we conclude with future work.

2 Background

The most commonly used word-cloud algorithm is the Wordle algorithm [7]. It positions words both horizontally and vertically and generally uses the word frequency to scale each word's size. When positioning the words, the algorithm uses a very compact representation of each word's outline, which ensures that words do not overlap and are spaced out as little as possible. This creates a very dense visualisation, but the use of vertical text and the placement of words within the outline of other words can make reading the visualisations harder for the user. In the algorithm presented here we thus only place text horizontally and also utilise more white-space between individual words.

In addition to this relatively common word-cloud layout, there have been a number of other layout approaches, including layouts for even less white-space [12,19], the ManiWordle approach [13], rolled-out word-clouds [21], or TagPies [10]. As will be described below, due to our focus on layout stability, we will use a very simple spiral-based layout algorithm. However, this could in the future be adapted to combine improved layouts with our high-performancy, stability-focused algorithm.

The main limitation motivating the work presented here is that these algorithms are static and are not designed to incorporate links between different word-clouds. Work on integrating temporal aspects has generally focused on approaches such as the SparkClouds [15] or Fish-Eye Clouds [23], where additional temporal information is integrated into the basic word-cloud layout. Alternatives have also used word-clouds as part of a more complex visual interface [24] or developed alternatives for comparing changes between two time-slices [6]. However, none of these address the requirement for adapting the actual word-cloud content over time.

[1] Locate, Explore, Retrace and Apprehend complex text variants https://lera.uzi.uni-halle.de

One of the earliest approaches for coordinated word-clouds was the Parallel Tag Clouds approach [4], which simplified the word-clouds' layout to a set of vertical columns, one per word-cloud. By using just one dimension in the layout algorithm and sorting the words alphabetically, Parallel Tag Clouds help the user in comparing multiple word-clouds, particularly since interactive visual clues were used to highlight the same word across all word-clouds. However, the word locations are not stable across the individual clouds.

Cui et al. [5] demonstrate taking one word-cloud's layout into account when generating the next in their context-preserving dynamic word-clouds. Their approach combines a time-line graph with multiple word-clouds. Each word-cloud uses a force-directed layout to factor in the layout from the previous word-cloud. This ensures that their word-clouds are semantically coherent and word locations are spatially relatively stable. However, the dynamic nature of the force-driven layout cannot ensure complete layout stability, only that the rough layout is preserved across word-clouds.

The Word Storm algorithm [3] produces coordinated word-clouds, by combining an iterative placement algorithm that averages word locations across all word-clouds with a gradient method inspired by multi-dimensional scaling that compacts the locations generated by the iterative algorithm. This creates coordinated word-clouds in which the word locations are very similar across the clouds. The clouds are then placed in a grid structure to allow for comparison. The core concept is expanded in the Word Flock algorithm [14], where not only exact words, but also semantic relatedness are used to place semantically similar words together. Instead of treating it as an optimisation problem, the COWORDS algorithm [20] uses a probabilistic sampling approach to pick a visual distribution that is both compact and retains stable word positions. While the approach produces word-clouds that are just as compact [1] as those in Word Storm, the probabilistic model allows for the integration of additional constraints, making the system more flexible.

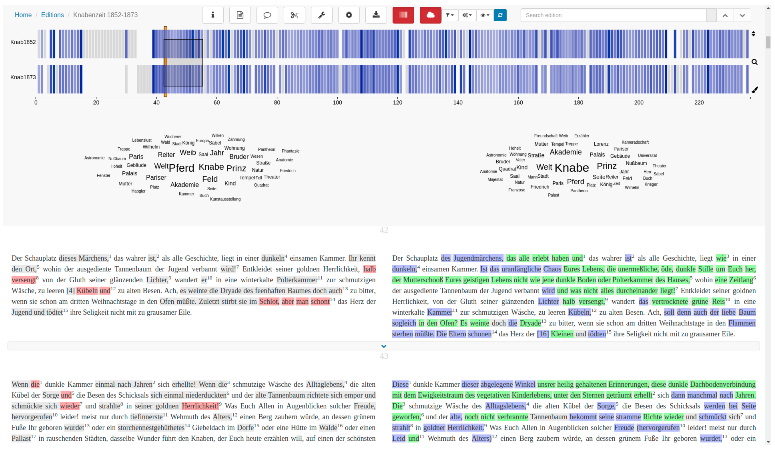
A different approach is taken in the ConcentriCloud visualisation [16], which visualises the word-clouds in a set of concentric circles. In the inner-most circle they place those words that appear in all word-clouds, while the outer circle has those words that only appear in specific word-clouds. The circles inbetween show words that appear in some, but not all word-clouds. This gives the user a good overview over the shared and distinguishing concepts. However, it also limits the number of word-clouds that can be compared, before the visualisation becomes unreadable.

All of these approaches are focused primarily on generating word-clouds to be shown together for comparison, a scenario in which small positional changes between the layouts do not present an issue. However, in our use scenario the system generates a large number of word-clouds that are shown sequentally and where even small positional changes between consecutive word-clouds generate a lot of visual noise, which would distract and potentially mask the actual changes. Additionally, as stated above, the user has full control over a large number of parameters (visualised segment size, which words to include, what statistics to use for the word sizes, number of words to visualise, ...), thus pre-calculation is not viable. The algorithm we developed is thus designed to create word-clouds where the word locations are completely fixed and which can be generated very quickly.

3 Project & Data

The work presented here was developed based on the requirements of the LERA tool. The tool is used to support humanities researchers in the inter-textual analysis of texts in a variety of contexts, with the aim of identifying similarities and differences between multiple versions of the same text. To do this the tool automatically segments and aligns the texts, the result of which can then be manually modified and corrected if so desired [2].

To analyse the texts the tool provides three linked visualisations (Figure 1): an overview visualisation that shows the identified segments and their alignment; the detailed text view that shows the actual text in each segment for all texts, and the word-clouds that visualise the relevant words in a section of the text. Multiple relevance metrics (frequency, Term-Frequency Inverse-Document-Frequency [TFIDF], ...) and filters (word length, part-of-speech tags, number of words to display, ...) can be applied to control what the word-clouds show.



Der Schauplatz dieses Märchens,[1] das wahrer ist,[2] als alle Geschichte, liegt in einer dunkeln[4] einsamen Kammer. Ihr kennt den Ort,[5] wohin der ausgediente Tannenbaum der Jugend verbannt wird![7] Entkleidet seiner goldnen Herrlichkeit, halb versengt[8] von der Gluth seiner glänzenden Lichter,[9] wandert er[10] in eine winterkalte Polterkammer[11] zur schmutzigen Wäsche, zu leeren [4] Kübeln und[12] zu alten Besen. Ach, es weinte die Dryade des feenhaften Baumes doch auch[13] zu bitter, wenn sie schon am dritten Weihnachtstage in den Ofen müßte. Zuletzt stirbt sie im Schlot, aber man schont[14] das Herz der Jugend und tödtet[15] ihre Seligkeit nicht mit zu grausamer Eile.

Der Schauplatz des Jugendmärchens, das alle erlebt haben und[1] das wahrer ist[2] als alle Geschichte, liegt wie[3] in einer dunkeln,[4] einsamen Kammer. Ist das uranfängliche Chaos Eures Lebens, die unermeßliche, öde, dunkle Stille um Euch her, der Mutterschooß Eures geistigen Lebens nicht wie jene dunkle Boden oder Polterkammer des Hauses,[5] wohin eine Zeitlang[6] der ausgediente Tannenbaum der Jugend verbannt wird und was nicht alles durcheinander liegt![7] Entkleidet seiner goldnen Herrlichkeit, von der Gluth seiner glänzenden Lichter halb versengt,[9] wandert das vertrocknete grüne Reis[10] in eine winterkalte Kammer[11] zur schmutzigen Wäsche, zu leeren Kübeln,[12] zu alten Besen. Ach, soll denn auch der liebe Baum sogleich in den Ofen? Es weinte doch die Dryade[13] zu bitter, wenn sie schon am dritten Weihnachtstage in den Flammen sterben müßte. Die Eltern schonen[14] das Herz der [16] Kleinen und tödten[15] ihre Seligkeit nicht mit zu grausamer Eile.

Wenn die[1] dunkle Kammer einmal nach Jahren[2] sich erhellte! Wenn die[3] schmutzige Wäsche des Alltaglebens,[4] die alten Kübel der Sorge und[5] die Besen des Schicksals sich einmal niederduckten[6] und der alte Tannenbaum richtete sich empor und schmückte sich wieder[7] und strahlte[8] in seiner goldnen Herrlichkeit![9] Was Euch Allen in Augenblicken solcher Freude, hervorgerufen[10] leider! meist nur durch tiefinnerste[11] Wehmuth des Alters,[12] einen Berg zaubern würde, an dessen grünem Fuße Ihr geboren wurdet[13] oder ein storchennestgehüthetes[14] Giebeldach im Dorfe[15] oder eine Hütte im Walde[16] oder einen Pallast[17] in rauschenden Städten, dasselbe Wunder führt den Knaben, der Euch heute erzählen will, auf einen der schönsten

Diese[1] dunkle Kammer dieser abgelegene Winkel unsrer heilig gehaltenen Erinnerungen, diese dunkle Dachbodenverbindung mit dem Ewigkeitstraum des vegetativen Kinderlebens, unter den Sternen geträumt erhellt[2] sich dann manchmal nach Jahren. Die[3] schmutzige Wäsche des Alltagslebens,[4] die alten Kübel der Sorge,[5] die Besen des Schicksals werden bei Seite geworfen,[6] und der alte, noch nicht verbrannte Tannenbaum bekommt seine stramme Richte wieder und schmückt sich[7] und strahlt[8] in goldner Herrlichkeit.[9] Was Euch Allen in Augenblicken solcher Freude (hervorgerufen[10] leider! meist nur durch Leid und[11] Wehmuth des Alters)[12] einen Berg zaubern würde, an dessen grünem Fuße Ihr geboren wurdet,[13] oder ein

Fig. 1. Screenshot of the LERA user-interface, showing the overview visualisation at the top with a range of segments selected, the word-clouds below, and two versions of the source text with differences highlighted at the bottom.

To visualise the words in the word-clouds, the user selects a range of segments in the overview and the word-clouds are immediately re-calculated for the selection range. The user can then move the selection range across the full length of the text and the word-clouds are updated in real-time to reflect the content of the newly selected range. This allows the researcher to develop an understanding of how the content of the text changes over the course of the text. In order to support the researcher in correctly interpreting the changes, it is necessary that the word locations remain stable as the selection range moves and words are added, removed, or change in importance. Otherwise after each step the researcher has to hunt for the changes.

The LERA system holds a number of different texts for analysis, the examples shown here are taken from a work by the nineteenth century author Karl Gutzkow "Aus der Knabenzeit". The text exists in two versions (1852 and 1873) consisting of 81695 and 75941 tokens, respectively, which after alignment produce 236 aligned segments, with the length of each segment ranging from single words to full paragraphs. For the word-cloud examples shown here we use the most relevant words identified via TFIDF and filtered to only include nouns. However, this filtering is undertaken before the data is passed to the word-cloud algorithm and the word-cloud algorithm does not make any specific demands of the tokens it is asked to visualise.

4 Stable Word-Clouds

To support the need for fast, stable word-clouds an initial naive approach was implemented, based on which the novel algorithm presented here was developed.

4.1 Naive stable word-clouds

An initial fast, but naive approach was developed based around the fixed spiral layout in Figure 2a. For the initial word-cloud the words are simply placed along the spiral, with each word's font size calculated from the word's importance, as in most word-cloud algorithms, and which depending on the user selection might be based on pure frequency, TFIDF score, or any of the other metrics provided by LERA. Then, as the user moves the selection, new words are placed into available slots on the spiral. The big question is at what point to free up those slots in the spiral for which the word that was originally placed there no longer occurs in the word-cloud for the current selection and is thus no longer shown. If no re-use is undertaken, then the spiral keeps expanding and the word-cloud becomes very sparse and hard to read.

To address this, the algorithm tracks how many steps the slot has remained empty. A step here is defined as the user moving the selection one segment. Thus if a slot has been empty for five steps, this means that the word that was originally placed there has not appeared as relevant in the last five segments. The problem lies in how to determine the number of steps s after which to flag a slot as free, so that it can be re-used with a different word. A low value drastically reduces the word-cloud's stability, as words first disappear and then re-appear in different locations, as their original slot has been re-used for a different word. At the same time a high value does not make good use of the available space, as spaces are not re-used for a long period of time.

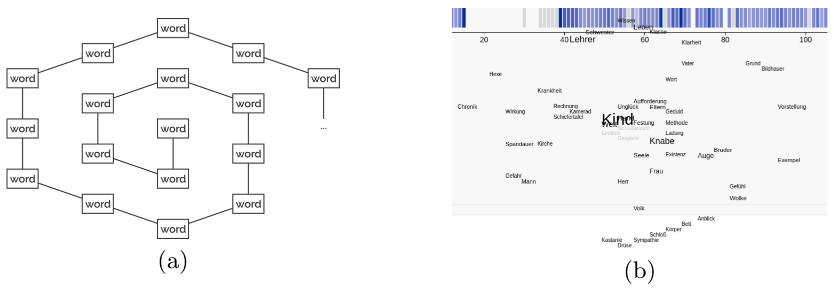


Fig. 2. a) Fixed spiral-based layout used to position words in the word-cloud. b) Example of a word-cloud that has degenerated into a very sparse display after the user has moved the selection range a few times.

Due to its naive nature, the algorithm's performance is very high and runs in the browser without any pre-processing. However, after extensive experimentation, we have concluded that the local-information approach is not able to provide the kind of stable and compact clouds (see Figure 2 for an example of how the word-clouds degenerate and become sparse) that are required.

4.2 Stable placement via pairwise orthogonality

As demonstrated by the existing work, the difficulty with finding stable word-cloud configurations is the large number of potential word placements, from which the most stable one needs to be selected. To circumvent this problem, the approach presented here is not based on finding stable word placements, but on determining which words never co-occur across the range of word-clouds that are generated. The words that never co-occur are grouped together and can then share the same location in the word-cloud. This allows us to re-use the word spiral for the final placement of words, but at the same time make it significantly more compact.

To determine which words can be grouped together the algorithm first constructs the word-segment occurrence matrix A, in which each row represents one aligned segment and each column one word. The matrix values are then set to the frequency with which a given word occurs in a given segment. Table 1 shows an example matrix for four segments and six words taken from the full matrix for the data-set. The columns are ordered by the increasing frequency with which the words occur over the whole text.

To determine which words can be grouped together we test which columns are pairwise orthogonal. Two columns a and b are orthogonal if their vector multiplication results in the zero vector (Equation 1) and each such pair is a candidate for grouping. In the example in Table 1 the columns “Herr” and “Baum”, “Herr” and “Meister”, “Herr” and “Gott”, “Baum” and “Kind”, “Baum” and “Knabe”, “Meister” and “Kind”, and “Meister” and “Knabe” are all pairwise orthogonal, because they do not share any rows where they have values greater than zero. Each of these pairs is thus a candidate for grouping together into a group.

Table 1. Example word-segment occurrence matrix A for the four segments with the words "Knabe, Kind Gott" (#1), "Gott, Gott, Meister, Baum" (#2), "Knabe" (#3), and "Knabe, Kind, Herr" (#4). In this example the user has selected to only include nouns in the word-cloud.

Segment	**Herr**	**Baum**	**Meister**	**Kind**	**Gott**	**Knabe**
#1	0	0	0	1	1	1
#2	0	1	1	0	2	0
#3	0	0	0	0	0	1
#4	1	0	0	1	0	1

$$a \cdot b^{\top} = \overrightarrow{0} \quad (1)$$

Determining which columns to group together to minimise the total number of groups is an NP-hard problem [11], thus we have developed the greedy Algorithm 1 to determine which columns to group together. The algorithm iterates over each column and, as long as the column has not already been assigned to a group, is added to a new group (lines 4–6). Then a new group occurrence vector is initialised with the current column's occurrences (line 7). Next the algorithm iterates over the remaining columns that have not yet been added to groups. Each of these columns is first checked for whether it has already been merged into another group (line 9) and then checked for orthogonality against the group vector (line 10). If the two are orthogonal, then the column is assigned to the current group, marked as merged, and the group's vector is updated by adding the column vector (lines 11–13). At the end of the outer loop each column has been assigned to a group and each group has at least one column in it.

For the example in Table 1 the first group would initially contain the column "Herr". Then the algorithm would iterate through the remaining columns and the first orthogonal column is "Baum", which would be added to the group. Next, the group would be compared to "Meister", but while "Herr" is orthogonal to "Meister", the merged group no longer is, thus neither "Meister" nor any of the other columns is not merged into the group. Table 2 shows how over four iterations the algorithm greedily merges together the columns into groups, adding the "Meister" and "Kind" group in the second iteration, while the words "Gott" and "Knabe" each get their own group as they are not orthogonal to each other.

The word-cloud visualisation generation then works based on the groups (Figure 3), rather than on individual words, but still uses the spiral to generate the placements. For the initial word-cloud the groups are placed along the spiral as in the naive approach, with individual words' font-size still determined by each word's relevance. The difference is that as the user moves the selection range, when a word is no longer displayed, because it no longer occurs in the selection range, that word's slot can immediately be re-used and re-filled with another word that has been allocated to the same group. This has two main benefits. First, as the number of groups is generally smaller than the number of

```
merged_columns = []
groups = []
for i in 0 .. |columns|
    if i not in merged_columns
        group = [i]
        merged_columns.push(i)
        group_v = columns[i]
        for j in i + 1 .. |columns|
            if j not in merged_columns:
                if orthogonal(group_v, columns[j])
                    merged_columns.push(j)
                    group.push(j)
                    group_v = group_v + columns[j]
                end
            end
        end
        groups.push(group)
    end
end
```

Algorithm 1: Greedy algorithm that constructs the merged groups based on the pairwise orthogonality of the individual columns. The algorithm's worst-case performance is $O(n^2 \cdot m)$ where n is the number of columns and m the number of rows.

Table 2. Example of the greedy Algorithm 1 processing the example data-set from Table 1. In each iteration one group is added, containing one or more word columns in that group. The fourth iteration represents the final set of groups to visualise.

Iteration	**Groups**	**Words Columns in the Group**
1	1	"Herr", "Baum"
2	1	"Herr", "Baum"
	2	"Meister", "Kind"
3	1	"Herr", "Baum"
	2	"Meister", "Kind"
	3	"Gott"
4	1	"Herr", "Baum"
	2	"Meister", "Kind"
	3	"Gott"
	4	"Knabe"

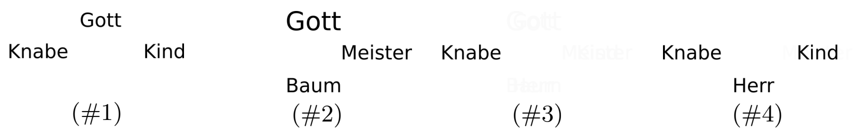


Fig. 3. Four word-clouds generated for the four segments in Table 1, but using the groups identified in Table 2 to place words on the spiral. Words in the same group appear in the same location, as can be seen when comparing the words “Kind” and “Meister” or “Herr” and “Baum”. Also visible is the scaling of the word “Gott” in the word-cloud for the second segment in which the word appears twice

words, overall the word-clouds are more compact. Second, because the number of groups is known before visualising, the visualisation as a whole can be scaled to ensure that the word-cloud is always visible as a whole, regardless of where the selection range is moved.

4.3 Multi-segment selection ranges

As the user is free to select ranges that cover multiple segments, the matrix A has to be updated to represent the multi-segment selection. In the updated matrix each row represents a series of segments and the matrix values are updated to show the frequency of each word across all selected segments. Table 3 shows the matrix A for a selection range of two segments. As the example shows, the number of rows is reduced by $|selectionrange| - 1$, in this case to 3 segments. This matrix is then fed into the grouping and layout algorithms as above.

Table 3. Example word-segment occurrence matrix A for the three two-segments-selected segments derived from the four original segments “Knabe, Kind Gott”, “Gott, Gott, Meister, Baum”, “Knabe”, and “Knabe, Kind, Herr”. The new segment #1 contains the words from the original segments #1 and #2, new segment #2 from the original segments #2 and #3, and new segment #3 from the original #3 and #4.

Segment	**Herr**	**Baum**	**Meister**	**Kind**	**Knabe**	**Gott**
#1	0	1	1	1	1	3
#2	0	1	1	0	1	2
#3	1	0	0	1	2	0

5 Evaluation

As stated above, the two core requirements for our algorithm were positional stability and high performance, both of which we evaluate here. A user-focused evaluation is planned in the context of evaluating the LERA system as a whole.

5.1 Stability of the Layout Algorithm

Looking at the positional stability, we undertook a qualitative and technical evaluation. For the qualitative we used ten-segment wide selection ranges and then visually inspected the results to verify that slots were being consistently filled and re-used. Figure 4 demonstrates the stability of the algorithm. For the two selection ranges the words “Hand”, “Prinz”, and “Tod” occur in both selections (highlighted in green) and stay in the same location. On the other hand the words “Fähnrich” and “Mäßigung” (highlighted in blue) are orthogonal and thus can share the same location.

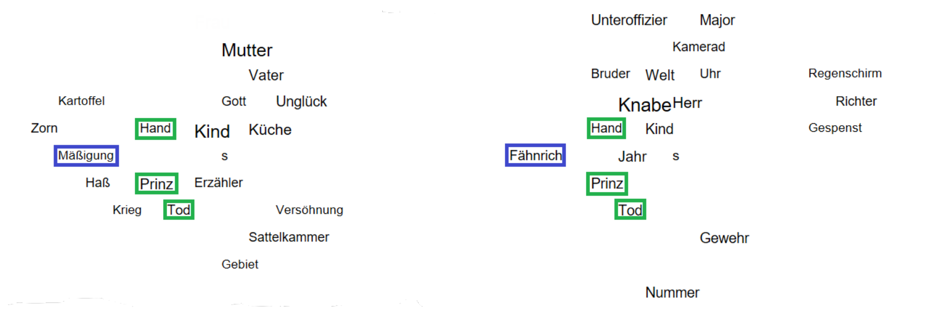


Fig. 4. Word-clouds showing two sets of 10 segments, demonstrating the stability of the positions for the words “Hand”, “Prinz”, and “Tod” (green) and the re-use of a position for the words “Mäßigung” and “Fähnrich” (blue).

Additionally we manually tested a wide range of selection ranges and then repeatedly moved the selection range across the text length. In our naive approach this would lead to the word-cloud degenerating as illustrated above (Figure 2b). In our tests the novel algorithm always maintained a stable cloud and compact word-cloud.

One recurring issue the testing identified is that the simple spiral layout in some cases has issues with word-overlap in the final display (Figure 5). This is because the spiral does not take into account the final, rendered size of the words and instead uses static distances between the word centres, which assume a relatively even size distribution. For future work there is thus a clear need to improve this by using an overlap-free layout algorithm.

5.2 Performance

To test the algorithm’s performance we automatically generated word-clouds for a range of selection sizes and measured the time needed for this. The algorithm was implemented in Ruby 2.4.5 and run on an Intel Core i7-3770 CPU with 16 GB of RAM running a Sabayon Linux with Kernel version 4.4.39. Table 4 shows the results from averaging ten runs of each configuration.

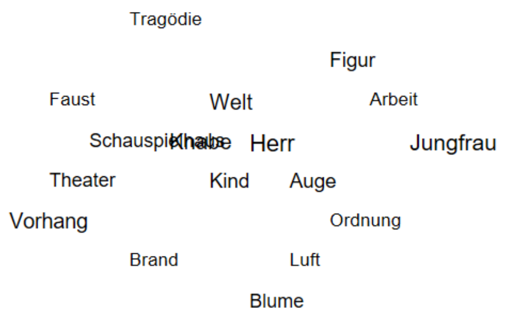


Fig. 5. Example of overlapping words in the final layout.

Table 4. Performance timings for a range of selection ranges. The number of groups is deterministic, but the run-time is reported as *average (standard deviation)* calculated from ten runs for each configuration displaying 100 words. The selection sizes have been determined based on feedback from LERA users.

Selection size	# Groups	Time
1	146	6.35s (0.18s)
40	151	4.11s (0.28s)
70	121	3.65s (0.13s)
90	117	3.74s (0.17s)

The main results in Table 4 clearly show that the algorithm's performance allows for running it on-demand without any pre-calculation. As is to be expected, the slowest run-time is for the single-segment word-clouds, where generating 146 groups from 2552 unique words takes approximately 6.35 seconds. For the largest practical selection size of 90 segments, run-time reduces to on average 3.74 seconds. The most common selection scenario is for a size of 40 and here the generation takes on average 4.11 seconds.

The other result is that through the orthogonality calculation the algorithm can substantially reduce the number of groups from 2552 unique input words to between 117 and 146 groups. As the examples show this number of groups can easily be visualised, but we are also investigating other space-filling curves [9] to create a denser layout that utilises all of the rectangular space and avoids overlaps.

As part of the performance testing we also assessed an optimisation based on the hypothesis that by reversing the direction of the inner loop (Algorithm 1, line 8–16), the overall performance would improve. The hypothesis behind the optimisation was that by starting with a column from the beginning of A, where the columns are generally sparse, and then trying to merge it with the more dense columns at the end of A, determining when two columns were not orthogonal would be faster, as it would take less row comparisons until a row was found where both columns had values.

Table 5 shows the results of running the algorithm with the inner loop direction reversed. It shows that for selection sizes of 1 and 40 it has little impact on the number of groups generated, but it does improve run time. In the case of a selection size of 1, the improvement is over 1 second. However, for the larger selection sizes, the performance is lower and it generates significantly more groups. The results confirm our basic hypothesis that merging very sparse with dense columns increases efficiency. However, when there are few sparse columns, as is the case for the larger selection sizes, by not letting the algorithm merge medium density columns first, the greedy approach is not as successful at reducing the total number of generated columns.

Table 5. Performance timings for a range of selection ranges and reversing the order of the inner loop. The number of groups is deterministic, but the time is reported as *average (standard deviation)* calculated from ten runs for each configuration displaying 100 words. The selection sizes have been determined based on feedback from LERA users.

Selection size	**# Groups**	**+/-**	**Time**	**+/-**
1	145	-1	$4.99s$ $(0.39s)$	$-1.36s$
40	153	+2	$4.03s$ $(0.53s)$	$-0.08s$
70	138	+17	$3.86s$ $(0.12s)$	$+0.21s$
90	128	+11	$3.73s$ $(0.12s)$	$+0.01s$

However, as the optimisation requires only minimal change to the algorithm, it is possible to dynamically change the inner loop's direction based on the selection size. While the exact breakpoint where the increased number of groups is no longer worth the time savings is likely to vary depending on the actual text, this can be determined automatically and the algorithm can then dynamically switch the inner loop's direction.

6 Conclusion & Future Work

Word-clouds are a useful tool for visualising text changes over time, but the generation of a series of word-clouds in which words remain in stable positions and which are nevertheless compact, is a highly time-consuming process. In this paper we have presented a novel word-cloud algorithm that uses word-occurance orthogonality to determine which words never cooccur and can thus share a location in the generated word-clouds. This compacts the clouds and allows the use of a simple spiral-based layout algorithm to create the a series of positionally stable word-clouds.

We have evaluated the algorithm and it successfully produces positionally stable series of word-clouds fast enough (between 4 and 6 seconds for the tested configurations) to allow for dynamic, online generation based on user selections. After this initial build time, the user can then instantaneously scroll through the generated word-clouds and easily see the changes over time.

For future work we propose further improvements in the performance and quality of the final word-clouds. While for the texts that we have used in the system dynamic generation is viable, for very long text further speed improvements are likely required, in particular for the orthogonality calculation. Two optimisations we are considering are the use of a heuristic that quickly determines whether it is at all possible for two columns to be independent, based on keeping track of the number of rows a column has values in, as if the sum of the number of non-zero rows of the two columns is greater than the total number of rows, the columns cannot be independent. The other planned optimisation is to use bitvectors to represent the columns, which would enable the use of fast bit-wise operations to check for orthogonality, rather than the current row-wise comparisons. Together these should allow the algorithm to scale to even very large texts.

As shown above, the layout algorithm currently has issues with word-overlap. To address this we plan to use a space-filling curve approach that takes into account the maximum size required for all the words in a group. This should only have a minimal impact on the generation time, but will further improve the ease with which the word-clouds can be interpreted and with which changes in the word-clouds can be visually tracked.

Finally, while the word-cloud algorithm has been deployed in the LERA system and has been used successfully by researchers, we have not yet formally evaluated its usability. In particular there are questions concerning whether colour- or animation-based cues are needed to indicate to users which words have appeared, been replaced, or changed in importance. We are planning to conduct such a user-study in the context of evaluating the LERA system as a whole.

References

1. Barth, L., Kobourov, S.G., Pupyrev, S.: Experimental comparison of semantic word clouds. In: International Symposium on Experimental Algorithms. pp. 247–258. Springer (2014)
2. Bremer, T., Molitor, P., Pöckelmann, M., Ritter, J., Schütz, S.: Zum einsatz digitaler methoden bei der erstellung und nutzung genetischer editionen gedruckter texte mit verschiedenen fassungen. In: Editio, vol. 29, pp. 29–51. Nutt-Kofoth, Rüdiger and Plachta, Bodo (2015). https://doi.org/10.1515/editio-2015-004, https://doi.org/10.1515/editio-2015-004
3. Castella, Q., Sutton, C.: Word storms: Multiples of word clouds for visual comparison of documents. In: Proceedings of the 23rd international conference on World wide web. pp. 665–676. ACM (2014)
4. Collins, C., Viegas, F.B., Wattenberg, M.: Parallel tag clouds to explore and analyze faceted text corpora. In: 2009 IEEE Symposium on Visual Analytics Science and Technology. pp. 91–98. IEEE (2009)
5. Cui, W., Wu, Y., Liu, S., Wei, F., Zhou, M.X., Qu, H.: Context preserving dynamic word cloud visualization. In: 2010 IEEE Pacific Visualization Symposium (PacificVis). pp. 121–128. IEEE (2010)
6. Dodds, P.S., Harris, K.D., Kloumann, I.M., Bliss, C.A., Danforth, C.M.: Temporal patterns of happiness and information in a global social network: Hedonometrics and twitter. PloS one **6**(12), e26752 (2011)

7. Feinberg, J.: Wordle. In: Steele, J., Iliinsky, N. (eds.) Beautiful Visualization Looking at Data through the Eyes of Experts. O'Reilly Media (2010)
8. Heimerl, F., Lohmann, S., Lange, S., Ertl, T.: Word cloud explorer: Text analytics based on word clouds. In: 2014 47th Hawaii International Conference on System Sciences. pp. 1833–1842. IEEE (2014)
9. Hilbert, D.: Über die stetige Abbildung einer Linie auf Flächenstück. Mathematische Annalen **38**, 459–460 (1891)
10. Jänicke, S., Blumenstein, J., Rücker, M., Zeckzer, D., Scheuermann, G.: Tagpies: Comparative visualization of textual data. In: VISIGRAPP (3: IVAPP). pp. 40–51 (2018)
11. Karp, R.M.: Reducibility among combinatorial problems. In: Complexity of computer computations, pp. 85–103. Springer (1972)
12. Kaser, O., Lemire, D.: Tag-cloud drawing: Algorithms for cloud visualization. arXiv preprint cs/0703109 (2007)
13. Koh, K., Lee, B., Kim, B., Seo, J.: Maniwordle: Providing flexible control over wordle. IEEE Transactions on Visualization and Computer Graphics **16**(6), 1190–1197 (2010)
14. Le, T., Lauw, H.W.: Word clouds with latent variable analysis for visual comparison of documents (2016)
15. Lee, B., Riche, N.H., Karlson, A.K., Carpendale, S.: Sparkclouds: Visualizing trends in tag clouds. IEEE transactions on visualization and computer graphics **16**(6), 1182–1189 (2010)
16. Lohmann, S., Heimerl, F., Bopp, F., Burch, M., Ertl, T.: Concentri cloud: Word cloud visualization for multiple text documents. In: 2015 19th International Conference on Information Visualisation. pp. 114–120. IEEE (2015)
17. Lohmann, S., Ziegler, J., Tetzlaff, L.: Comparison of tag cloud layouts: Task-related performance and visual exploration. In: IFIP Conference on Human-Computer Interaction. pp. 392–404. Springer (2009)
18. Schütz, S., Pöckelmann, M.: Lera - explorative analyse komplexer textvarianten in editionsphilologie und diskursanalyse. In: Digital Humanities im deutschsprachigen Raum 2016 - Konferenzabstracts. pp. 249–253. Leipzig (2016), http://dhd2016.de/boa.pdf
19. Seifert, C., Kump, B., Kienreich, W., Granitzer, G., Granitzer, M.: On the beauty and usability of tag clouds. In: 2008 12th International Conference Information Visualisation. pp. 17–25. IEEE (2008)
20. Silva e Silva, L.G., Assunção, R.M.: Cowords: a probabilistic model for multiple word clouds. Journal of Applied Statistics **45**(15), 2697–2717 (2018)
21. Strobelt, H., Spicker, M., Stoffel, A., Keim, D., Deussen, O.: Rolled-out wordles: A heuristic method for overlap removal of 2d data representatives. In: Computer Graphics Forum. vol. 31, pp. 1135–1144. Wiley Online Library (2012)
22. Vuillemot, R., Clement, T., Plaisant, C., Kumar, A.: What's being said near "martha"? exploring name entities in literary text collections. In: 2009 IEEE Symposium on Visual Analytics Science and Technology. pp. 107–114. IEEE (2009)
23. Wang, J., Dent, K.D., North, C.L.: Fisheye word cloud for temporal sentiment exploration. In: CHI'13 Extended Abstracts on Human Factors in Computing Systems. pp. 1767–1772. ACM (2013)
24. Wanner, F., Jentner, W., Schreck, T., Stoffel, A., Sharalieva, L., Keim, D.A.: Integrated visual analysis of patterns in time series and text data-workflow and application to financial data analysis. Information Visualization **15**(1), 75–90 (2016)