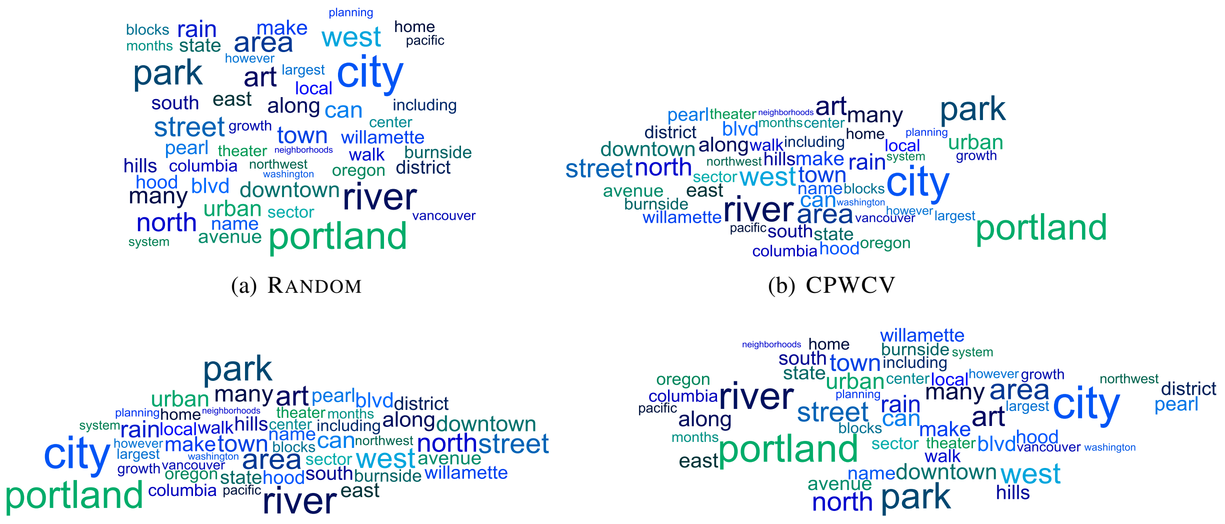
An Experimental Study of Algorithms for Semantics-Preserving Word Cloud Layout

Lukas Barth, Stephen G. Kobourov, Sergey Pupyrev
Department of Computer Science
University of Arizona

Abstract

We study the problem of computing semantic-preserving word clouds in which semantically related words are close to each other. We implement three earlier algorithms for creating word clouds and three new ones. We define several metrics for quantitative evaluation of the resulting layouts such as realized adjacencies, layout distortion, compactness, and uniform area use. We then compare all algorithms according to these metrics, using two different data sets of word documents from Wikipedia and ALENEX papers. We show that two of our new algorithms, based on extracting heavy subgraphs from a weighted graph, outperform all the others by placing many more pairs of related words so that their bounding boxes are adjacent. Moreover, this improvement is not achieved at the expense of significantly worsened measurements for the other metrics (distortion, compaction, uniform area use). The online system implementing the algorithms, all source code, and data sets are available at `http://wordcloud.cs.arizona.edu`.



(a) RANDOM (b) CPWCV (c) SEAM CARVING (d) STAR FOREST

Figure 1: Word clouds for the Wikipedia page "Portland" (top 50 words).

1 Introduction

Word clouds and tag clouds are often used to visually summarize text. The practical tool, Wordle [25], with its high quality design, graphics, style and functionality popularized word cloud visualizations as an appealing way to summarize the content of a webpage, a research paper, or a political speech. While tools like this are popular [14, 21, 26], most of them, including Wordle itself, have a potential shortcoming: They do not visualize the relationships between the words in any way, as the placement of the words is completely independent of their context. At the same time, someone looking at a word cloud can not help but perceive two words that are placed next to each other as being related in some way; see Fig. 1(a). While some of the more recent word cloud visualization tools aim to incorporate semantics in the layout [4, 13, 18, 27], none provide any guarantees about the quality of the layout in terms of semantics. The semantic-preserving word cloud problem is related to classic graph layout problems [12, 23], where the goal is to draw graphs so that vertex labels are readable and Euclidean distances between pairs of vertices are proportional to the underlying graph distance between them. Typically, vertices are treated as points and label overlap removal is a post-processing step [5, 10, 11].

A more formal model of the problem, using a vertex-weighted and edge-weighted graph [1], asks to explicitly measure how many related words are placed next to each other. The vertices in the graph are the words in the document, with weights corresponding to some measure of importance (e.g., word frequency). The edges in the graph capture the semantic relatedness between pairs of words (e.g., co-occurrence), with weights corresponding to the strength of the relation. Each vertex must be drawn as a rectangle with fixed dimensions and with area determined by its weight. A contact between two rectangles is a common boundary, and if two rectangles are in contact, we say that these rectangles *touch*. The edge weight between two vertices corresponds to the value of making the corresponding rectangles touch. The *realized adjacencies* value is the sum of the edge weights for all pairs of touching rectangles. The goal is to find a representation of the given rectangles, which maximizes the realized adjacencies.

This model is related to work in rectangle representations of graphs, vertices are axis-aligned rectangles with non-intersecting interiors and edges correspond rectangles with non-zero length common boundary. Every graph that can be represented this way is planar and every triangle in such a graph is a facial triangle. These two conditions are also sufficient to guarantee a rectangle representation [2, 9, 20, 22, 24]. In a recent survey Felsner [8] reviews many rectangulation variants, including squarings.

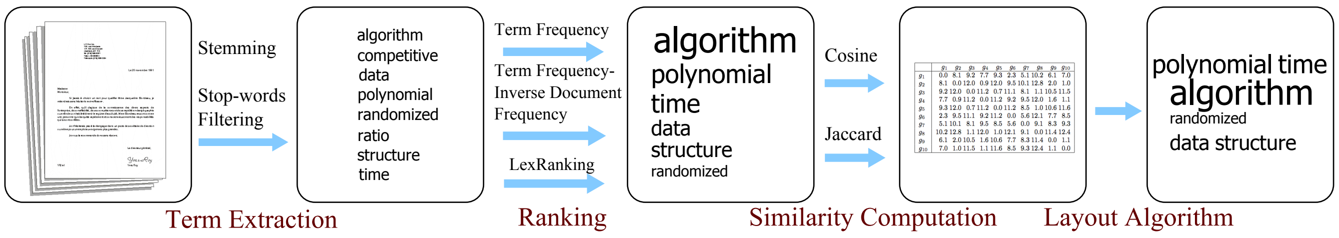


Figure 2: Overview of creating a semantic word cloud visualization.

In this paper we define metrics that aim to quantitatively measure the more abstract goal of "semantic preservation": realized adjacencies, distortion, compactness, and uniform area utilization. Then we implement and extensively test six algorithms for generating word clouds: three earlier methods and three new ones. Two of the new algorithms outperform the rest in terms of realized adjacencies, while not negatively impacting any of the remaining metrics. An online system implementing all the algorithms, and which also provides all source code and all data sets is at `http://wordcloud.cs.arizona.edu`.

2 Experimental Setup

All the algorithms for producing word clouds take as input an edge-weighted graph and rectangles of fixed dimensions associated with every vertex. There are several parameters to consider in a preprocessing step needed to extract this information from input texts. Our preprocessing pipeline is illustrated in Fig. 2.

Term Extraction: We first split the input text into sentences, which are then tokenized into a collection of words using the open-source toolkit [28]. Common stop-words such as "a", "the", "is" are removed from the collection. The remaining words are grouped by their stems using the Porter Stemming Algorithm [19], so that related words such as "dance", "dancer", and "dancing" are reduced to their root, "danc". The most common variation of the word is used in the final word cloud.

Ranking: In the next step we rank the words in order of relative importance. We use three different ranking functions, depending on word usage in the input text. Each ranking function orders words by their assigned weight (rank), and the top n of them are selected, where n is the number of words visualized in the final word cloud. *Term Frequency (tf)*, is the most basic ranking function and one used in most traditional word cloud visualizations. Even after removing common stop-words, term frequency tends to rank highly many semantically meaningless words. *Term Frequency-Inverse Document Frequency (tf-idf)* addressed this problem by normalizing the frequency of a word by its frequency in a larger text collection. In our case $\mathrm{tf}-\mathrm{idf}(w,d,D) = \mathrm{tf}(w,d) \times \mathrm{idf}(w,D)$, where w is the word, d is the current input text and D is the collection of all our text documents. Our third ranking function is based on the LexRank algorithm [7], which was already used for creating semantic-preserving word clouds by Wu *et al.* [27]. The algorithm is a graph-based method for computing relative importance of textual units using eigenvector centrality. In the graph $G = (V, E)$, vertices represent words and edges represent co-occurrence of the words in the same sentences. A weight w_{ij} of an edge (i, j) is the number of times word i appears in the same sentence as word j. From the graph it is possible to determine rank values of the words as described in [7].

Similarity Computation: Given the ranked list of words, we calculate an $n \times n$ matrix of pairwise similarities so that related words receive high similarity values. We use two similarity functions depending on the input text. The *Cosine Similarity* between words i and j can be computed as $sim_{ij} = \frac{w_i \cdot w_j}{||w_i|| \cdot ||w_j||}$, where $w_i = \{w_{i1}, \ldots, w_{in}\}$ and $w_j = \{w_{j1}, \ldots, w_{jn}\}$ are the vectors representing co-occurrence of the words with other words in the input text. The *Jaccard Similarity* coefficient is an alternative measure for calculating pairwise word similarity. Let S_i be a set of sentences in which the word i appears. Jaccard coefficient is the number of sentences two words appeared together in divided by the number of sentences either word appeared in: $sim_{ij} = \frac{|S_i \cap S_j|}{|S_i \cup S_j|}$. In both cases for all pairs of words, the similarity function produces a value between 0, indicating that a pair of words is not related, and 1, indicating that words are very similar.

3 Word Cloud Layout Algorithms

Here we briefly describe three early and three new word cloud layout algorithms, all of which we implemented and tested extensively. The input for all algorithms is a collection of n rectangles, each with a fixed height and width proportional to the rank of the word, together with $n \times n$ matrix with entries $0 \leq sim_{ij} \leq 1$. The output is a set of non-overlapping positions for the rectangles. The first algorithm, RANDOM, is used in the Wordle tool. The CPWCV and SEAM CARVING are the semantics-preserving word clouds algorithms recently considered in the literature. The INFLATE is our new heuristic, which is naturally extends CPWCV. The remaining two heuristics, STAR FOREST and CYCLE COVER, are the novel approaches based on approximation algorithms proposed in [1].

3.1 Wordle (Random)

The Wordle algorithm places one word at a time [25]. The algorithm uses greedy strategy, aiming to use space as efficiently as possible. First the words are sorted by weight (proportional to the height of the corresponding rectangle in our setting) in decreasing order. Then for each word in the order, a position is picked at random. If the word intersects one of the previously placed words then it is moved along a spiral of increasing radius radiating from its starting position. Although the original Wordle has many variants (e.g., words can be horizontal, vertical, or at arbitrary orientation), we always place words horizontally.

3.2 Context-Preserving Word Cloud Visualization (CPWCV)

The algorithm of Cui *et al.* [4] attempts to capture semantics building a word cloud in two steps. First a dissimilarity matrix Δ is computed, where $\Delta_{ij} = 1 - sim_{ij}$ for every pair of words i, j represents ideal distances between words in n-dimensional space. Multidimensional scaling (MDS) is performed to obtain two-dimensional positions for the words so that the given distances are (approximately) preserved. Since the step usually creates a very sparse layout, in the second step compaction is achieved via a modified force-directed algorithm. Attractive forces between pairs of words reduce empty space, while repulsive ensure that words do not overlap. An additional force attempts to preserver semantic relations between words. To this end, a triangular mesh (Delaunay triangulation in the implementation) is computed from the initial word positions, and the additional force attempts to keep the mesh planar.

3.3 Seam Carving

The algorithm of Wu *et al.* [27] uses seam carving, a content-aware image resizing technique, to keep semantically similar words close to each other. Here a preliminary overlap-free word layout is computed, using a force-directed algorithm adapted from CPWCV [4]. Then the screen space is divided into regions, and for each region an energy function is computed. A connected left-to-right or top-to-bottom path of low energy regions is called a seam. The major step of the algorithm is iteratively carving out seams of low energy to remove empty space between words. Since the order of removing seams greatly affects the final result, a dynamic programming approach is used to find an optimal order. This gives us a word cloud in which no further seam can be removed.

3.4 Inflate-and-Push (INFLATE)

We designed and implemented an alternative simple heuristic method for word layout, which aims to preserve semantic relations between pairs of words. The heuristic starts with scaling down all word rectangles by some constant $S > 0$ (in our implementation $S = 100$) and computing MDS on dissimilarity matrix Δ in which $\Delta_{ij} = \frac{1-sim_{ij}}{S}$. At this point, the positions of the words respect their semantic relations; that is, semantically similar words are located close to each other. We then iteratively increase dimensions of all rectangles by 5%

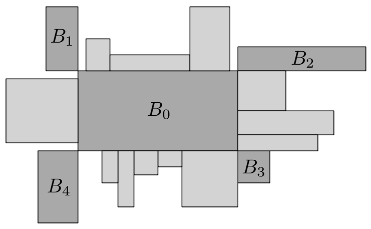
("inflate" words). After each iteration some words may overlap. We resolve the overlaps using the repulsive forces from the force-directed model of the CPWCV algorithm ("push" words). Since the dimensions of each rectangle grows by only 5%, the forces generally preserve relative positions of the words. In practice, 50 iterations of the "inflate-push" procedure is sufficient for constructing a compact layout of words with their original dimensions.

3.5 Star Forest

A star is a tree of depth at most 1, and a star forest is a forest whose connected components are all stars. Our algorithm has three steps. First we partition the given graph, obtained from the dissimilarity matrix Δ, into disjoint stars (extracting a star forest). Then we build a word cloud for every star (realizing a star). Finally, the individual solutions are packed together to produce the final word cloud. Each step is described in details next.

We extract stars from the given graph greedily. We find a vertex v with the maximum adjacent weight, that is, the one for which $\sum_{u \in V} sim(v,u)$ is maximized. We then treat the vertex as a star center and the vertices $V \setminus \{v\}$ as leaves. A set of words that will be adjacent to star center v is computed, and these words are removed from the graph. This processes is repeated with the smaller graph, until the graph is empty.

The problem of computing the best set of words to be adjacent to a star center v is similar to the Knapsack problem defined as follows: Given a set of items, each with a size and a value, pack a maximum-valued subset of items into a knapsack of given capacity. Let B_0 denote the box corresponding to the center of the star; see Fig. 3(a). In any optimal solution there are four boxes B_1, B_2, B_3, B_4 whose sides contain one corner of the center box B_0 each. Given B_1, B_2, B_3, B_4, the problem reduces to assigning each remaining box B_i to at most one of the four sides of B_0, which completely contains the contact between B_i and B_0. The task of assigning boxes to a side of B_0 is naturally converted to an instance of Knapsack: The size of a box B_i is its width for the horizontal sides and its height for the vertical sides of B_0, the value is the edge weight between B_i and B_0. Now we run the algorithm for the Knapsack problem for the top side of B_0, remove the realized boxes, and proceed with the bottom, left, and then right sides of the central box. In order to solve Knapsack, we use the polynomial-time approximation scheme described in [15].



(a) Solving a star

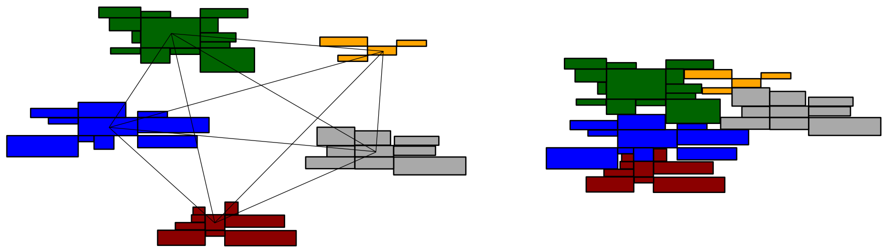

(b) Packing star solutions

Figure 3: Star Forest algorithm.

Finally, the computed solutions for individual stars are packed together aiming for a compact drawing, which preserves the semantic relations between words in different stars. We begin by placing the stars on the plane so that no pair of words overlap; see Fig. 3(b). For every pair of stars s_1, s_2, we compute its similarity as the average similarity between the words comprising s_1 and s_2, that is, $sim(s_1, s_2) = \frac{\sum_{v \in s_1} \sum_{u \in s_2} sim(u,v)}{|s_1||s_2|}$. MDS is utilized to find the initial layout with $k(1 - sim(s_1, s_2))$ being the ideal distance between the pair of stars. In our implementation, we set the scaling factor $k = 10$ to ensure an overlap-free result. Then a force-directed algorithm is employed to obtain a compact layout. Note that the algorithm adjusts the positions of whole stars rather than individual words; thus, the already realized adjacencies between words are preserved. The algorithm utilizes attractive forces aiming at removing empty space and placing semantically similar words close to each other. The force between the stars is defined as $f_a(s_1, s_2) = k_a(1 - sim(s_1, s_2))\Delta l$, where Δl

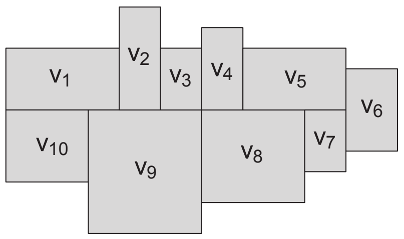
represents the minimum distance between two central rectangles of the stars. The repulsive force is used to prevent overlaps between words. The force only exists if two words occlude each other. It is defined as $f_r(s_1, s_2) = k_r min(\Delta x, \Delta y)$, where Δx (Δy) is the width (height) of the overlapping region. In experiments, we found that the priorities of the forces $k_a = 15, k_r = 500$ work well. As in a classical force-directed scheme, the algorithm iteratively adjust positions of the stars. We impose a limit of 500 iterations, although the process converges very quickly.

3.6 Cycle Cover

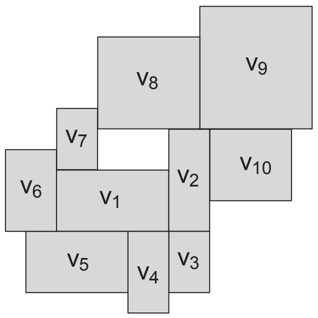
This algorithm is also based on extracting a heavy planar subgraph from the graph $G = (V, E)$ defined by the similarity matrix. In particular, finding a cycle cover (vertex-disjoint set of cycles) with maximum weight, and realizing all cycles in the cover by touching boxes is a $\frac{2}{d_{max}+1}$ approximation algorithm for total realized adjacencies, for G with maximum degree d_{max}.

Although the heaviest cycle cover can be found in polynomial time in the size of the graph (see Chapter 10 in [16]), the algorithm is too slow in practice as it requires computation of a maximum weighted matching in a not necessarily bipartite graph. We use the following simpler algorithm instead, which transforms G into a related bipartite graph H. We create $V(H)$ by first copying all vertices of G. Then for each vertex $v \in V(G)$, we add a new vertex $v' \in V(H)$, and for each edge $(u, v) \in E(G)$, we create two edges $(u', v) \in E(H)$ and $(u, v') \in E(H)$ with weights $sim(u, v)$. The resulting graph H is bipartite by construction, thus making it easy to compute a maximum weight matching in H. The matching induces a set of vertex-disjoint paths and cycles in G since every vertex u is matched with one v' and every u' is matched with one v.

Once cycles and paths are extracted, we place the corresponding boxes so that all edges are realized as follows. For a given cycle $(v_1, v_2, \ldots, v_n)$, let t be maximum index such that $\sum_{i \le t} w_i < \sum_{i \le n} w_i/2$, where w_i is the width of the i-th word. Vertices $v_1, v_2, \ldots, v_t$ are placed side by side in order from left to right with their bottom sides aligned on a shared horizontal line, while vertices $v_n, v_{n-1}, \ldots, v_{t+2}$ are placed from left to right with their top sides aligned on the same line; see Fig. 4(a). It remains to place v_{t+1} in contact with v_t and v_{t+2}, which can be done by adding v_{t+1} to the side of minimum width, or straddling the line in case of a tie. It is possible that the resulting layout has poor aspect ratio (as cycles can be long), so we convert cycles with more than 10 vertices into paths by removing the lightest edge. When realizing the edges of a path, we start with words v_1 and v_2 placed next to each other. The i-th word is added to the layout so that it touches v_{i-1} using its first available side in clockwise order radiating from the side of the contact between v_{i-2} and v_{i-1}. This strategy tends to create more compact, spiral-like layouts; see Fig. 4(b).



(a) Realizing a cycle



(b) Realizing a path

Figure 4: CYCLE COVER algorithm.

In the final step, the computed solutions for individual cycles and paths are packed together, aiming for a compact drawing which preserves the semantic relations between words in different groups. We use the same force-directed algorithm (described in the previous section for STAR FOREST) for that task.

4 Metrics for Evaluating Word Cloud Layouts

While a great deal of the world cloud appeal is *qualitative* and depends on good graphic design, visual appeal, etc., we concentrate of *quantitaive* metrics that capture several desirable properties. We use these metrics to evaluate the quality of the six algorithms under consideration. The metrics are defined so that the measurement is a real number in the range $[0, 1]$ with 1 indicating the best value and 0 indicating the worst one.

Realized Adjacencies: Our primary quality criterion for semantics-preserving algorithms is the total realized adjacency weight. In practice, proper contacts are not strictly necessary; even if two words do not share a non-trivial boundary, they can be considered as "adjacent" if they are located very close to each other. We assume that two rectangles touch each other if the distance between their boundaries is less than 1% of the width of the smaller rectangle. For each pair of touching rectangles, the weight of the edge is added to the sum. We normalize the sum by dividing it by the sum of all edge weights, thus measuring the fraction of the adjacency weight that was realized. Hence, the metric *Realized Adjacencies* is defined by $\alpha = \frac{\sum_{(u,v)\in E_{realized}} sim(u,v)}{\sum_{(u,v)\in E} sim(u,v)}$. Note that this value is always less than 1 as the input graph (as described in Section 2) is a complete graph and thus highly non-planar. On the other hand, the contact graph of rectangles is always planar, which means that in most cases it is impossible to realize contacts between all pairs of words.

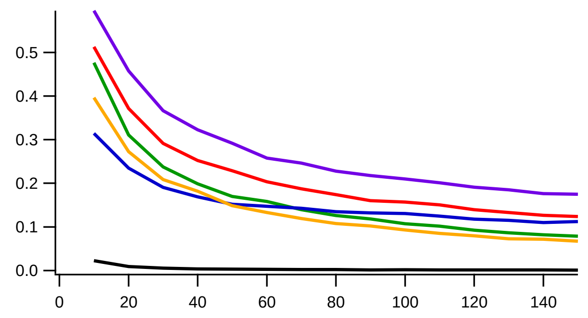
Distortion: This metric is used to measure another aspect of how well the desired similarities are realized. Instead of just looking at the weights of the adjacencies that are realized by touching rectangles, as in the metric *Realized Adjacencies*, *Distortion* measures whether the distances between all pairs of rectangles are proportional to the desired dissimilarities of the words. In order to compute the metric for a given layout, we construct a matrix of ideal distances between the words with entry $\Delta_{uv} = 1 - sim(u, v)$ for words u and v, and a matrix of actual distances between words in which an entry d_{uv} denotes the distance between the words u and v. Unlike in traditional measurements for a distortion of a graph embedding, we modify the definition of distance to reflect the use of non-zero area boxes for vertices, instead of points. Note that even for touching rectangles (and thus, naturally considered as close to each other as possible), the distance between their centers may be quite large. Hence, we measure the distance between two words as the minimal distance between any pair of points on the corresponding two rectangles. We then consider the matrices as two random variables and compute the correlation coefficient between them:

$$r = 1 - \frac{\sum_{(u,v)\in E}(\Delta_{uv} - \overline{\Delta})(d_{uv} - \overline{d})}{\sqrt{\sum_{(u,v)\in E}(\Delta_{uv} - \overline{\Delta})^2 \sum_{(u,v)\in E}(d_{uv} - \overline{d})^2}},$$

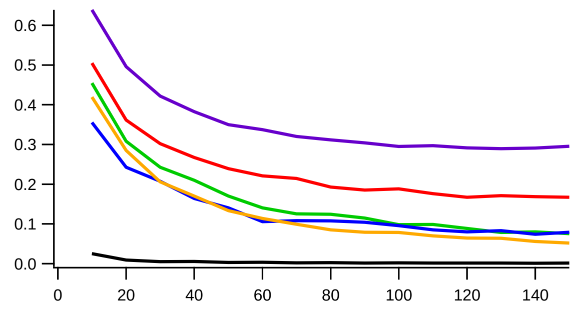
where $\overline{\Delta}$ and $\overline{d}$ are the average values of the corresponding distances. Since the correlation coefficient takes values between -1 and 1, *Distortion* is defined by $\beta = (r + 1)/2$. The value $\beta = 1$ indicates a perfect correspondence between dissimilarities and distances, while $\beta = 0.5$ means that the values are independent.

Compactness: This metric measures the compactness of the produced word cloud. Ideally, one wants to minimize the total used area and to avoid wasted space. Starting with the rectangles, we first compute the *used area* as the area covered by words, by adding up the areas of all the rectangles. Clearly, any layout needs at least that much space as overlaps are not allowed in our setting. We then compute the *total area* of the layout. To this end, we consider two approaches. In the first one the total area is the area of the bounding box containing all rectangles, while in the second approach, the area of the convex hull of all rectangles constitutes the total area. The *Compactness* metric is therefore $\gamma = 1 - \frac{\text{used area}}{\text{total area}}$, with value 1 corresponding to the best possible packing.

Uniform Area Utilization: Arguably, a desired property of a word cloud is the randomness of the distribution of the words in the layout. In highly non-random distributions some parts of the cloud are densely populated while others are sparse. The metric captures the uniformity of the word distribution. In order to compute the



(a) WIKI dataset, TF-IDF Ranking, Cosine Similarity



(b) ALENEX dataset, Lex Ranking, Jaccard Similarity

Figure 5: Realized adjacencies for word clouds of various size.

metric for a word cloud with n words, we divide the drawing into $\sqrt{n} \times \sqrt{n}$ cells. Then for each rectangle, find indices (x, y) of the cell containing the center of the rectangle. We may now consider the words as a discrete random variable, and measure its information entropy, or the amount of uncertainty. To this end, we compute the relative entropy of the variable (induced by a given layout of words) with respect to the uniform distribution [3]: $H = \sum_{i,j} p(i,j) \log \frac{p(i,j)}{q(i,j)}$, where the sum is taken over the created cells, $p(i,j)$ is the actual number of words in the cell (i,j), and $q(i,j) = 1$ is the number of words in the cell in the uniformly distributed word cloud. Since the entropy is maximized as $\log n$, *Uniform Area Utilization* is defined by $\delta = 1 - \frac{H}{\log n}$, where the value of 1 corresponds to an "ideal" word cloud.

Aspect Ratio and Running Time: While not formal evaluation metric for word cloud visualization, we also measure the aspect ratio of the final layouts and the running times of the algorithms. We use a standard measurement for the *Aspect Ratio* of the word clouds: W/H, where W and H are the width and height of the bounding box. The running time of word cloud algorithms is an important parameter as many such tools are expected to work in real-time and delays of more than a few seconds would negatively impact their utility. We measure *Running Time* for the execution of the layout phase of the evaluated algorithms, excluding the time needed to parse text, compute similarities between words, and draw the result on the display.

5 Results and Discussion

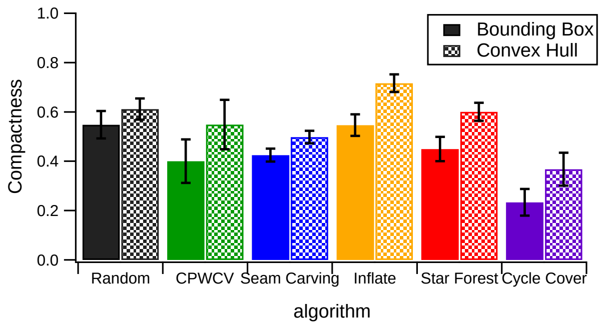
We evaluate all algorithms on two datasets: (1) WIKI, a set of 112 plain-text articles extracted from the English Wikipedia, each consisting of at least 200 words, and (2) ALENEX, a set of 45 research papers published in ALENEX in 2010-2012. The texts are preprocessed using the pipeline described in Section 2. Every algorithm is executed 5 times on each text; hence, the results present average values of metrics over the runs. Our implementation is written in Java, and the experiments were performed on a machine with an Intel i5 3.2GHz processor with 8GB RAM.

Realized Adjacencies: The CYCLE COVER algorithm outperforms all other algorithms on all tested instances; see Fig. 5 (and Fig. 11 and Fig. 12 in the Appendix). Depending on the method for extracting pairwise word similarities, the algorithm realizes 30-50% of the total edge weights for small graphs (up to 50 words) and $15 - 30\%$ for larger graphs. It is worth noting here that CYCLE COVER improves upon the best existing heuristic by a factor of nearly 2. The STAR FOREST also realizes more adjacencies than the existing semantics-aware methods. However, for large word clouds (with over 100 words) the difference between STAR FOREST and the existing algorithms drops to $2 - 5\%$.

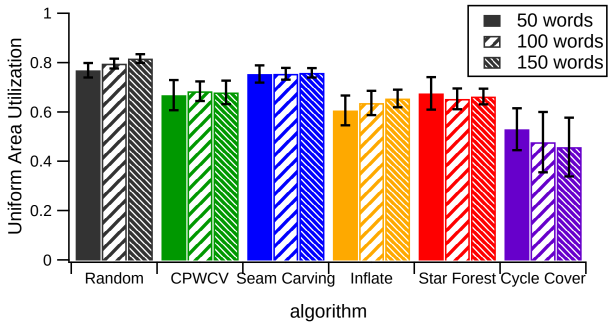
It is noticeable that the CPWCV algorithm and our INFLATE algorithm perform similarly in all settings. This is the expected behavior, as both methods are based on similar force-directed models. For instances with up to $80 - 100$ words, these techniques are better than the SEAM CARVING algorithm. On the other hand, the more sophisticated SEAM CARVING algorithm is preferable for larger word clouds, which confirms earlier results [27]. Not surprisingly, the RANDOM algorithm realizes only a small fraction of the total weight as it is not designed for preserving semantics between words.

Compactness: We measure compactness in terms of both bounding box and convex hull of the drawing; see Fig. 6(a). We first observe that the results in general do not depend on the used dataset, the ranking algorithm, and the way similarities are computed. The word clouds constructed by the INFLATE algorithm are the most compact, while the remaining algorithms perform very similar. In practice, such "overcompacted" drawings are not very desirable since adjacent words are difficult to read; see Fig. 14(a). In order to alleviate this, when generating actual word clouds we increase the dimensions of each rectangle by $10 - 20\%$ and run the layout algorithm for the new instance. In the new drawings, the words are easy to read, and the area is still used efficiently; see Fig. 14(b).

Uniform Area Utilization: As expected, the RANDOM algorithm generates word clouds with the most uniform word placement; see Fig. 6(b). SEAM CARVING also utilizes area uniformly. The remaining methods are all mostly comparable, with CYCLE COVER being the worst. The standard deviation for these 4 algorithms is relatively high, which means that some of the created word clouds may be rather non-uniform. Similar to the *Compactness* metric, we do not observe any significant difference of area utilization for different setting (dataset, ranking and similarity algorithms).



(a) WIKI, TF-IDF Ranking, Jaccard Similarity

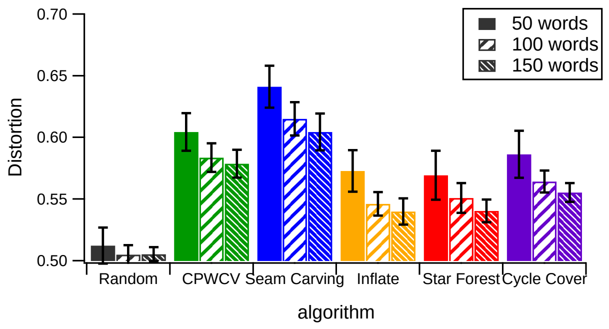


(b) ALENEX, Lex Ranking, Cosine Similarity

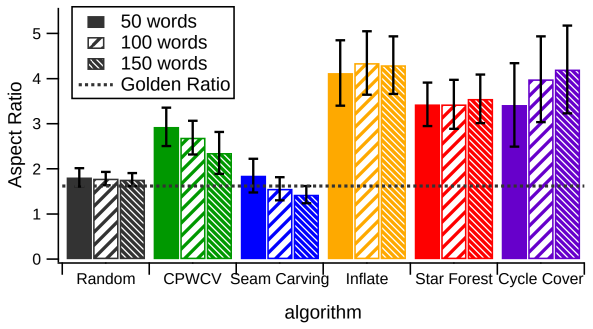
Figure 6: (a) Mean and standard deviation of *Compactness* for texts with 100 words. (b) Mean and standard deviation of *Uniform Area Utilization* for word clouds of various size.

Distortion: The SEAM CARVING algorithm steadily produces the most undistorted layouts (note as usual, although a bit counterintuitive here, high values are good); Fig. 7(a). Not surprisingly, the correlation coefficient between the dissimilarity matrix and the actual distance matrix produced by RANDOM is very close to 0 (hence, the value of *Distortion* is 0.5). For the remaining algorithms the results are slightly worse, but comparable. Note that all considered algorithms (except RANDOM) have a common feature: they start with an initial layout for the words (or for clusters of words) constructed by multidimensional scaling. At that point, the *Distortion* measurements are generally very good, but the layout is sparse. Next the algorithms try to compact the drawing, and the compaction step worsens *Distortion* significantly; see Fig. 15. Hence, there is an inherent tradeoff between compactness of a drawing and its distortion.

Aspect Ratio and Running Time: RANDOM and SEAM CARVING produce word clouds with aspect ratio close to the golden ratio ($\frac{1+\sqrt{5}}{2}$), which is commonly believed to be aesthetically pleasing; see Fig. 7(b). The CPWCV and the STAR FOREST algorithms generate drawings with the aspect ratio from $5:2$ to $7:2$, and the measurements are mostly independent of the number of words in a word cloud. The results of CYCLE COVER are sometimes wider due to the nature of the algorithm. We emphasize here that none of the considered al-



(a) WIKI, TF Ranking, Cosine Similarity

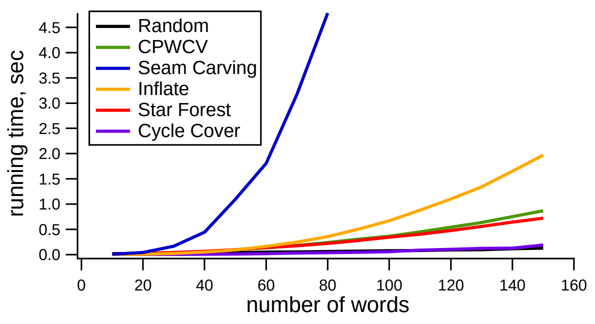


(b) ALENEX, TF Ranking, Cosine Similarity

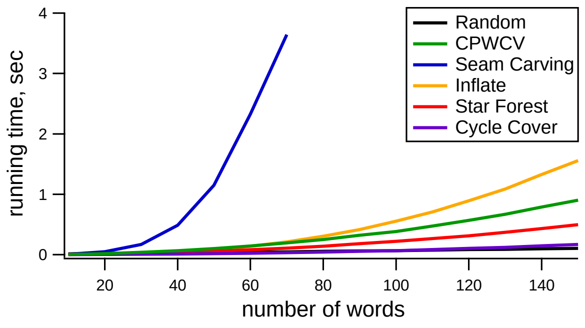
Figure 7: (a) Measurements of *Distortion*. Note that the y-axis starts at value 0.5. (b) *Aspect Ratio*

gorithms is designed to preserve a specific aspect ratio. If this turns out to be a major aesthetic parameter, optimizing the layout algorithms, while maintaining the desired aspect ratio might be a meaningful direction for future work.

The running times of all but one algorithm are reasonable, dealing with $100 - 150$ words within 2 seconds; see Fig. 8. The exception here is the SEAM CARVING algorithm which requires $15 - 25$ seconds per graph on the ALENEX dataset and $30 - 50$ on the WIKI dataset.



(a) WIKI, TF Ranking, Cosine Similarity

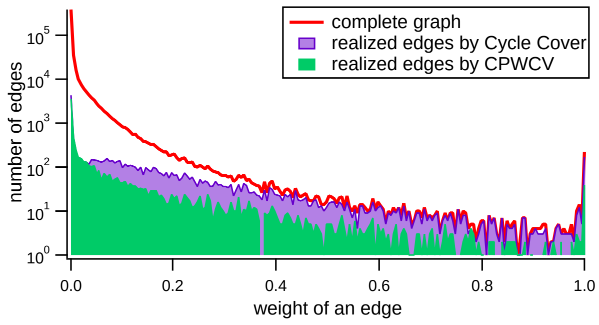


(b) ALENEX, TF-IDF Ranking, Jaccard Similarity

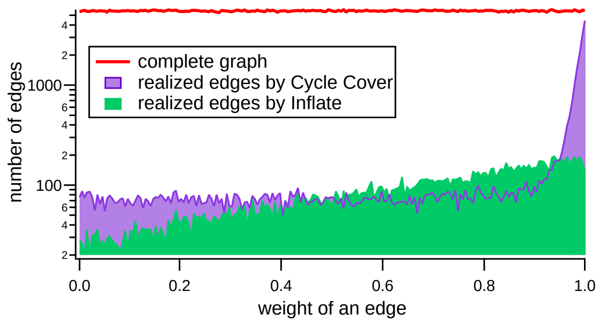
Figure 8: Running time of the algorithms for word clouds of various size.

Discussion: The CYCLE COVER and the STAR FOREST algorithms are better at realizing adjacencies, and they are comparable to the other algorithms in compactness, area utilization, and distortion. A likely explanation is that the distribution of edge weights is highly non-uniform for real-world texts and articles; see Fig. 9(a). The weights follow a Zipf distribution with many light edges and few heavy ones. Further, a small fraction of the edges (about 8% for WIKI and about 5% for ALENEX datasets) carry half of the total edge weight. It is known that words in text documents follow this distribution (Chapter 5 in [17]) and that might explain why pairs of co-related words behave similarly. Both CYCLE COVER and STAR FOREST are based on the same principle: first extract “heavy” subgraphs and then realize contacts within each subgraph independently. On the other hand, the existing semantics-preserving approaches try to optimize all contacts simultaneously by considering the complete graph. Our new strategy is more effective as it realize most of the heavy edges; see Fig. 9(a). It is possible that in a different scenario (where edge weights are not similarities between words) the distribution may be close to uniform; even in such a scenario CYCLE COVER and STAR FOREST outperform existing methods, but the fraction of the realized edge weights is much smaller; see Fig. 9(b) and Fig. 13.

None of the existing semantics-preserving algorithms make any guarantees (or approximation guarantees) about the optimality of realized edge weights when the input graph is complete (as in real-world examples).



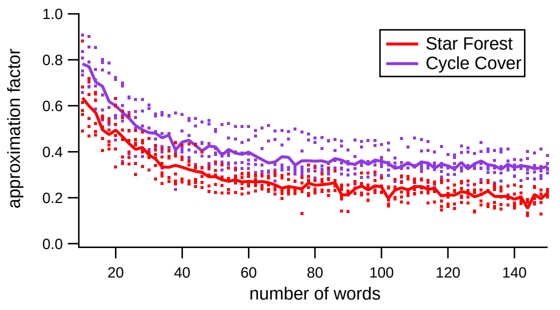
(a)



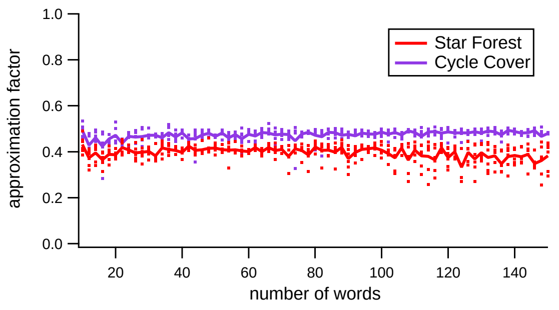
(b)

Figure 9: Distribution of weights (similarities between pairs of words) among edges (red line), and the fraction of realized edges with a given weight (closed regions) for a new and an existing algorithms. (a) A graph constructed for a real-world text (ALENEX dataset). (b) A complete graph in which edge weights are randomly chosen between 0 and 1.

However, it is still interesting to analyze how well these two algorithms realize adjacencies. Note that the sum of all edge weights in a graph is not a good upper bound for an optimal solution since the realized subgraph is necessarily planar and thus contains at most $3n-6$ edges. Instead, we compute a maximum weight spanning tree of the graph. Since the weight w of the tree is a $1/3$-approximation for the maximum planar subgraph of G [6], the value of $3w$ is also an upper bound for the maximum realized edge weights. On average CYCLE COVER produces results which are at most 3 times less than the optimum for graphs with 150 vertices; see Fig. 10.



(a) ALENEX, TF Ranking, Cosine Similarity



(b)

Figure 10: Comparing the realized weight to the upper bound for maximum realized edge weights. Dots represent single instances, solid lines represent average values over 5 runs. (a) Similarities constructed for a real-world text. (b) A graph with random weights between words.

6 Conclusions and Future Work

We quantitatively evaluated six algorithms for computing word cloud layouts. The RANDOM algorithm uses available area in the most uniform way and creates the most compact drawings, but does not place semantically related words close to each other. The SEAM CARVING algorithm produces layouts with low distortion and good aspect ratio, but they are not compact and the algorithm is very time-consuming. CPWCV and the new INFLATE algorithms perform very similarly in our experiments, even though INFLATE is much simpler. The two new algorithms STAR FOREST and CYCLE COVER, based on extracting heavy subgraphs, outperform all other methods in terms of realized adjacencies and running time, and they are competitive in the other metrics. We hope to find an algorithm with guaranteed approximation factor for extracting heavy planar subgraphs from general graphs, which would lead to a guaranteed fraction of realized adjacencies. We also want to consider more relaxed versions of adjacency (two words might not have rectangles which share a border, but there is no other word between them), and other usability issues.

References

[1] L. Barth, S. Kobourov, S. Pupyrev, and T. Ueckerdt. On Semantic Word Cloud Representation. `http://arxiv.org/abs/1304.8016`, 2013.

[2] A. L. Buchsbaum, E. R. Gansner, C. M. Procopiuc, and S. Venkatasubramanian. Rectangular layouts and contact graphs. *ACM Transactions on Algorithms*, 4(1), 2008.

[3] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley-Interscience, 2006.

[4] W. Cui, Y. Wu, S. Liu, F. Wei, M. Zhou, and H. Qu. Context-preserving dynamic word cloud visualization. *Computer Graphics and Applications, IEEE*, 30(6):42–53, 2010.

[5] T. Dwyer, K. Marriott, and P. J. Stuckey. Fast node overlap removal. In *13th Symposium on Graph Drawing*, pages 153–164, 2005.

[6] M. Dyer, L. Foulds, and A. Frieze. Analysis of heuristics for finding a maximum weight planar subgraph. *European Journal of Operational Research*, 20(1):102–114, 1985.

[7] G. Erkan and D. R. Radev. Lexrank: graph-based lexical centrality as salience in text summarization. *J. Artif. Int. Res.*, 22(1):457–479, 2004.

[8] S. Felsner. Rectangle and square representations of planar graphs. In *Thirty Essays on Geometric Graph Theory*, pages 213–248. Springer, 2013.

[9] É. Fusy. Transversal structures on triangulations: A combinatorial study and straight-line drawings. *Discrete Mathematics*, 309(7):1870–1894, 2009.

[10] E. R. Gansner and Y. Hu. Efficient, proximity-preserving node overlap removal. *Journal of Graph Algortihms and Applications*, 14(1):53–74, 2010.

[11] E. R. Gansner, Y. Hu, and S. C. North. Interactive visualization of streaming text data with dynamic maps. *Journal of Graph Algortihms and Applications*, 17(4):515–540, 2013.

[12] M. Kaufmann and D. Wagner. *Drawing graphs: methods and models*, volume 2025. Springer, 2001.

[13] K. Koh, B. Lee, B. H. Kim, and J. Seo. Maniwordle: Providing flexible control over Wordle. *IEEE Trans. Vis. Comput. Graph.*, 16(6):1190–1197, 2010.

[14] B. Y. Kuo, T. Hentrich, B. M. Good, and M. D. Wilkinson. Tag clouds for summarizing web search results. In *Proceedings of the 16th international conference on World Wide Web*, pages 1203–1204, 2007.

[15] E. L. Lawler. Fast approximation algorithms for knapsack problems. *Mathematics of Operations Research*, 4(4):339–356, 1979.

[16] L. Lovász and M. Plummer. *Matching Theory*. Akadémiai Kiadó, Budapest, 1986.

[17] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[18] A. Nocaj and U. Brandes. Organizing search results with a reference map. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2546–2555, 2012.

[19] M. F. Porter. An algorithm for suffix stripping. *Program: Electronic Library & Information Systems*, 40(3):211–218, 1980.

[20] P. Rosenstiehl and R. E. Tarjan. Rectilinear planar layouts and bipolar orientations of planar graphs. *Discrete & Computational Geometry*, 1(1):343–353, 1986.

[21] C. Seifert, B. Kump, W. Kienreich, G. Granitzer, and M. Granitzer. On the beauty and usability of tag clouds. In *12th IEEE Conference on Information Visualisation*, pages 17–25, 2008.

[22] C. Thomassen. Interval representations of planar graphs. *Journal of Combinatorial Theory, Series B*, 40(1):9–20, 1986.

[23] I. Tollis, P. Eades, G. Di Battista, and L. Tollis. *Graph drawing: algorithms for the visualization of graphs*. Prentice Hall, New York, 1998.

[24] P. Ungar. On diagrams representing graphs. *J. of the London Math. S.*, 28:336–342, 1953.

[25] F. B. Viégas, M. Wattenberg, and J. Feinberg. Participatory visualization with Wordle. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1137–1144, 2009.

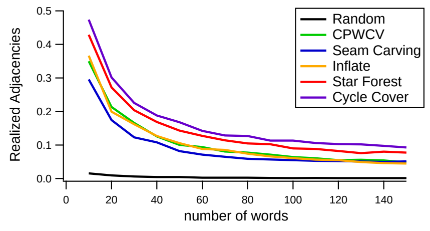
[26] F. B. Viegas, M. Wattenberg, F. Van Ham, J. Kriss, and M. McKeon. Manyeyes: a site for visualization at internet scale. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1121–1128, 2007.

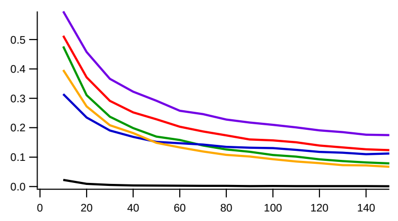
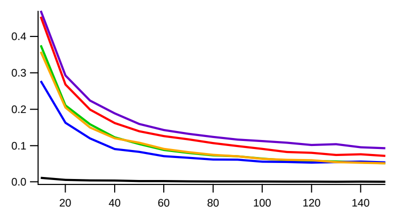
[27] Y. Wu, T. Provan, F. Wei, S. Liu, and K.-L. Ma. Semantic-preserving word clouds by seam carving. In *Computer Graphics Forum*, volume 30, pages 741–750, 2011.

[28] Apache OpenNLP. `http://opennlp.apache.org`.

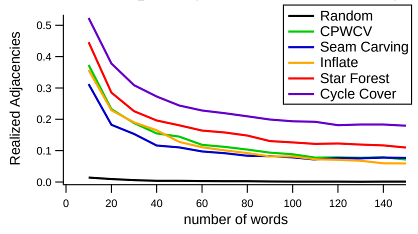
Appendix

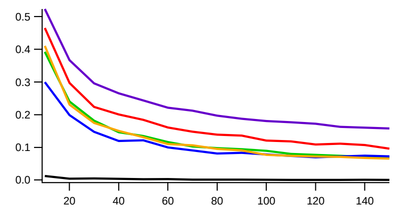
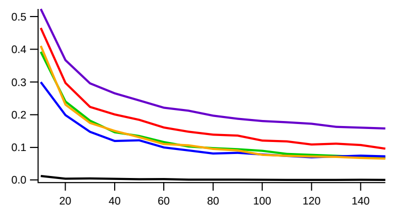
Here we include more examples of word clouds generated by the six algorithms discussed in the paper. We also include several additional plots, showing the effect of different term ranking and similarity matrix computations on the realized adjacencies metric.



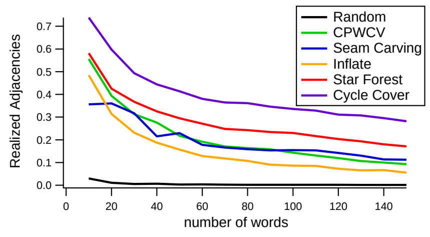
Term Frequency, Cosine Similarity TF-IDF, Cosine Similarity Lex Ranking, Cosine Similarity

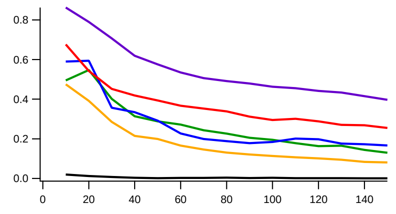
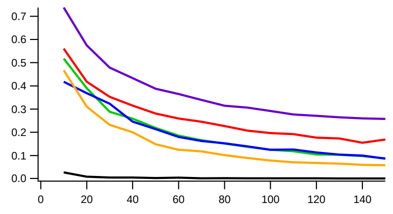


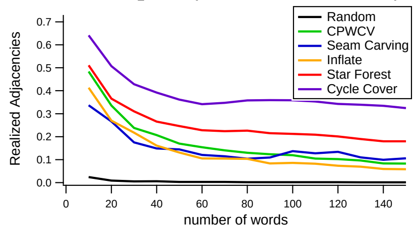
Term Frequency, Jaccard Similarity TF-IDF, Jaccard Similarity Lex Ranking, Jaccard Similarity

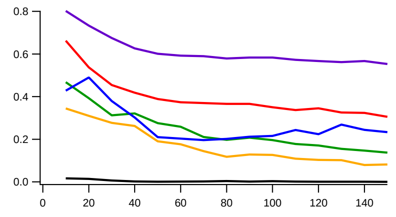
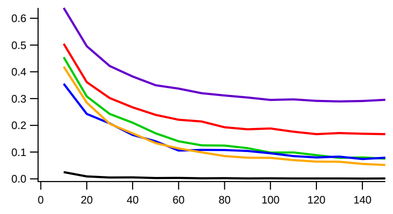
Figure 11: Realized adjacencies for the WIKI dataset.



Term Frequency, Cosine Similarity TF-IDF, Cosine Similarity Lex Ranking, Cosine Similarity



Term Frequency, Jaccard Similarity TF-IDF, Jaccard Similarity Lex Ranking, Jaccard Similarity

Figure 12: Realized adjacencies for the ALENEX dataset.

Acknowledgements: We acknowledge Dagstuhl Seminar 12261, where the problem of theoretically-sound semantic-preserving word cloud algorithms originated. In particular, we thank Steve Chaplick, Sara Fabrikant, Priya Gopalakrishnan, Anna Lubiw, Martin Nöllenburg, Yoshio Okamoto, Shiv Pande, Günter Rote, Claudio Squarcella, Alexander Wolff, and Ben Yee for productive discussions about this problem.

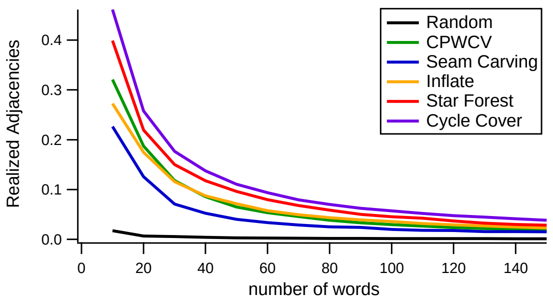
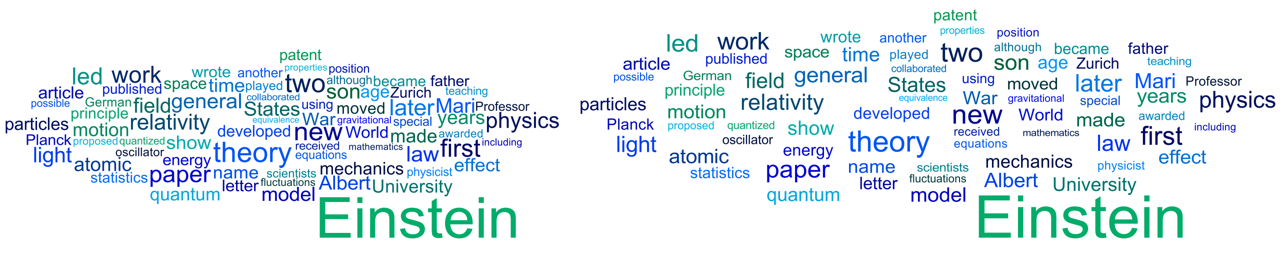
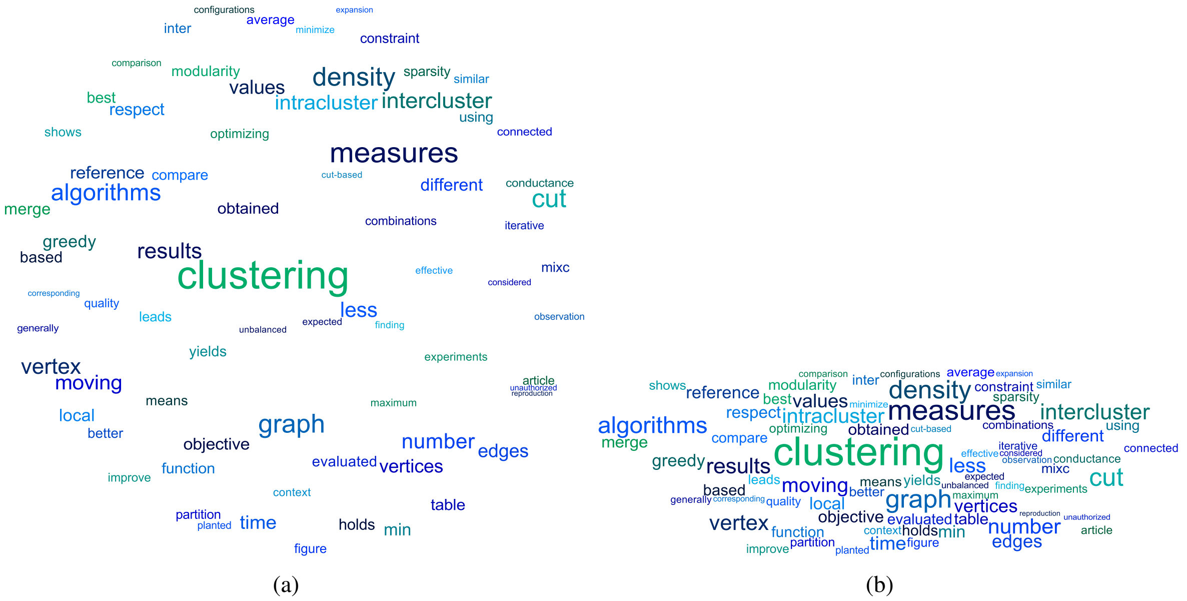


Figure 13: Realized adjacencies for a complete graph in which edge weights are randomly chosen between 0 and 1.



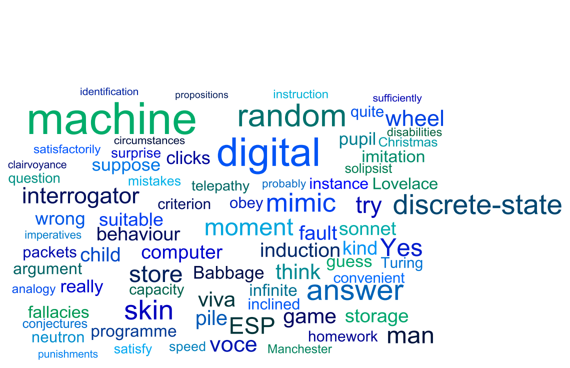
(a) (b)

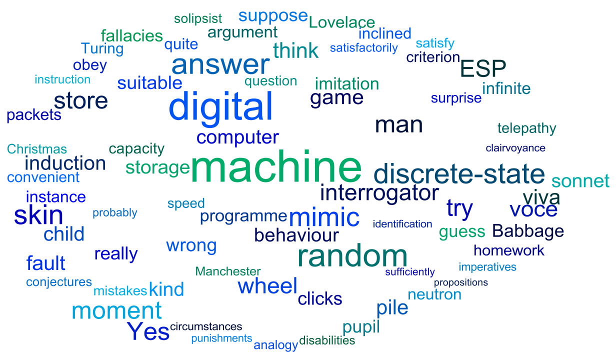
Figure 14: Wikipedia article “Albert Einstein”. (a) Overcompacted word cloud constructed with the INFLATE algorithm. (b) Result after increasing rectangles by 20%.

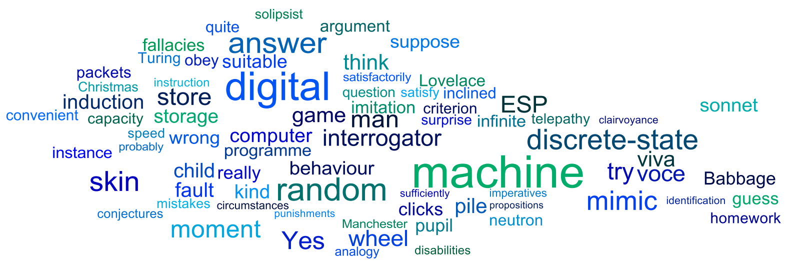


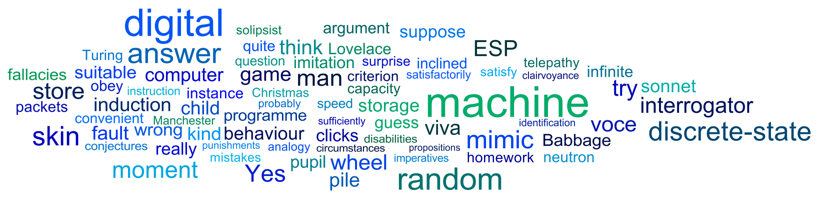
(a) (b)

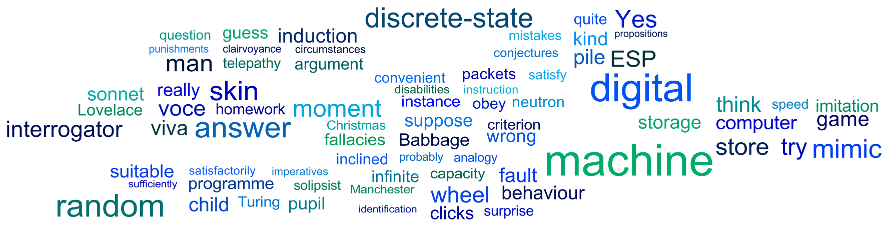
Figure 15: ALENEX paper “Experiments on Density-Constrained Graph Clustering” by Görke *et al.* (a) Initial layout found by MDS with *Distortion* $\beta = 0.68$. (b) Final result constructed by CPWCV with worsened value of $\beta = 0.57$.


(a) Random: $\alpha = 0.01, \beta = 0.5, \gamma = 0.58, \delta = 0.82$


(b) Seam Carving: $\alpha = 0.12, \beta = 0.59, \gamma = 0.47, \delta = 0.79$


(c) CPWCV: $\alpha = 0.15, \beta = 0.55, \gamma = 0.63, \delta = 0.76$


(d) Inflate: $\alpha = 0.15, \beta = 0.54, \gamma = 0.75, \delta = 0.72$


(e) Star Forest: $\alpha = 0.23, \beta = 0.53, \gamma = 0.56, \delta = 0.7$

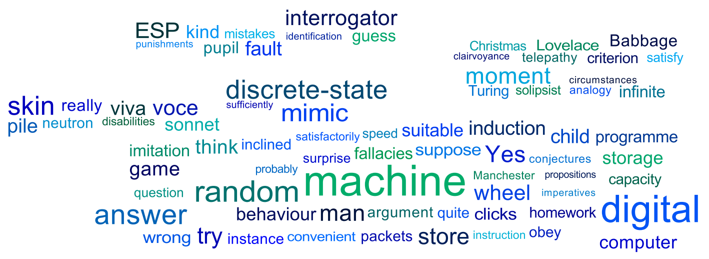

(f) Cycle Cover: $\alpha = 0.35, \beta = 0.56, \gamma = 0.61, \delta = 0.61$

Figure 16: "Computing Machinery and Intelligence" by A. M. Turing (top 75 words).