Experimental Comparison of Semantic Word Clouds

Lukas Barth, Stephen G. Kobourov, Sergey Pupyrev

Department of Computer Science, University of Arizona, USA

Abstract. We study the problem of computing semantics-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. Then the algorithms are compared according to these metrics, using two data sets of documents from Wikipedia and research papers. We show that two of our new algorithms 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.

1 Introduction

In the last few years, word clouds have become a standard tool for abstracting, visualizing, and comparing text documents. For example, word clouds were used in 2008 to contrast the speeches of the US presidential candidates Obama and McCain. More recently, the German media used them to visualize the 2013 coalition agreement and compare it to a similar agreement from 2009. Word clouds, or their close relatives tag clouds, are often used to represent importance among items (e.g., bands popularity on Last.fm) or serve as a navigation tool (e.g., Google search results).

A practical tool, Wordle [18], with its high quality design, graphics, style and functionality popularized word cloud visualizations as an appealing way to summarize the text. While tools like this are popular and widely used [19], most of them, including Wordle itself, have a potential shortcoming: they do not capture the relationships between the words in any way, as word placement is independent of context. But humans, as natural pattern-seekers, cannot help but perceive two words that are placed next to each other in a word cloud as being related in some way. In linguistics and in natural language processing if a pair of words often appears together in a sentence, then this is seen as evidence that this pair of words is linked semantically [12]. Thus, when using a word cloud, it makes sense to place such related words close to each other; see Fig. 1. In fact, recent empirical studies show that semantically clustered word clouds provide improvements over random layouts in specific tasks such as searching, browsing, and recognition [6, 17]. Finally, semantic word clouds had higher user satisfaction compared to other layouts [20].

Nearly all recent word cloud visualization tools aim to incorporate semantics in the layout [5, 10, 15, 21]. However, none provide any guarantees about the quality of the layout in terms of semantics. The existing algorithms are usually based on force-directed graph layout heuristicsto add such functionality. In contrast, we propose several new algorithms with such performance guarantees. Consider the following natural formal model of semantics-preserving word cloud visualization, based on a vertex-weighted and edge-weighted graph [1, 2]. The vertices in the graph are the words in

(a) RANDOM (b) STAR FOREST

Fig. 1. Word clouds for the Wikipedia page "Copenhagen" (top 50 words).

the document, with weights corresponding to some measure of importance (e.g., word frequency). The edges 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 an axis-aligned rectangle (box) with fixed dimensions determined by its weight. A contact between two boxes is a common boundary, and if two boxes are in contact, these boxes are called *touching*. The *realized adjacencies* is the sum of the edge weights for all pairs of touching boxes. The goal is a representation of the given boxes, maximizing the realized adjacencies.

This model is related to 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 [3, 9].

In this paper we first define metrics that 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. The online system implementing all the algorithms, and which also provides all source code and all data sets is available 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 Apache OpenNLP. 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 [16], 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 have 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 shown in the word cloud. *Term Frequency* (TF), is the most basic ranking function and one used in most traditional

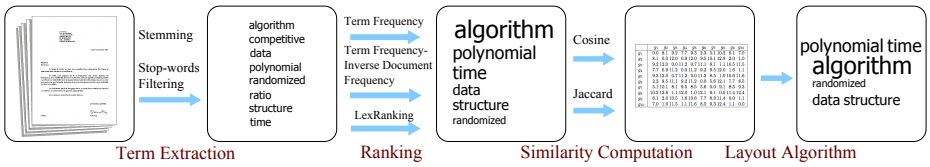


Fig. 2. Overview of creating a semantic word cloud visualization.

word cloud visualizations. Even after removing common stop-words, however, TF tends to rank highly many semantically meaningless words. *Term Frequency-Inverse Document Frequency* (TF-IDF) addresses this problem by normalizing the frequency of a word by its frequency in a larger text collection. In our case TF-IDF$(w, d, D) =$ TF$(w, d) \times$ 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 uses the *LexRank* algorithm [8], a graph-based method for computing relative importance of words, and already used for semantic-preserving word clouds [21]. 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. The rank values are computed using eigenvector centrality in G.

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 the number of sentences two words appeared together in divided by the number of sentences either word appeared in: $sim_{ij} = \frac{|S_i \cap S_j|}{|S_i \cup S_j|}$, where S_i is the set of sentences containing word i. In both cases 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 width and height 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,

3.1 Wordle (Random)

The Wordle algorithm places one word at a time in a greedy fashion, aiming to use space as efficiently as possible [18]. First the words are sorted by weight (proportional to the height of the corresponding rectangle) 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, mixed), we always place words horizontally.

3.2 Context-Preserving Word Cloud Visualization (CPWCV)

The algorithm of Cui *et al.* [5] aims to capture semantics in two steps. First a dissimilarity matrix Δ is computed, where $\Delta_{ij} = 1 - sim_{ij}$ represents ideal distances between

words i and j 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, the second step compacts the layout via a force-directed algorithm. Attractive forces between pairs of words reduce empty space, while repulsive forces ensure that words do not overlap. An additional force attempts to preserve 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.* [21] uses seam carving, a content-aware image resizing technique, to capture semantics. Here a preliminary overlap-free word layout is computed, using a force-directed algorithm adapted from CPWCV [5]. 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 to iteratively carve out seams of low energy to remove empty spaces between words. Since the order of seam removal greatly affects the final result, a dynamic programming approach is used to find an optimal order. The final result is a word cloud in which no further seam can be removed.

3.4 Inflate-and-Push (INFLATE)

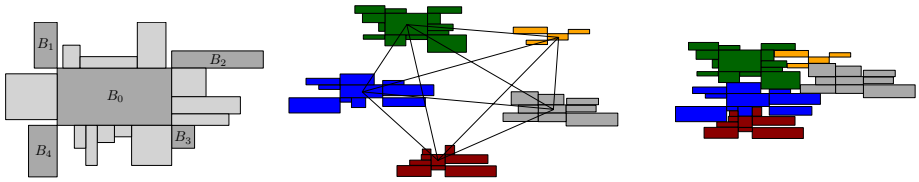
We designed and implemented an alternative simple heuristic method for word layout, which aims to preserve semantic relations between pairs of words. The heuristic starts by scaling down all word rectangles by some constant $S > 0$ (in our implementation $S = 100$) and computing MDS on dissimilarity 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 the 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 suffice.

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 result. The steps are described 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.

Selecting the best set of words to be adjacent to a star center v is related to the Knapsack problem, where given a set of items, each with a size and a value, we want



(a) Solving a star (b) Packing star solutions

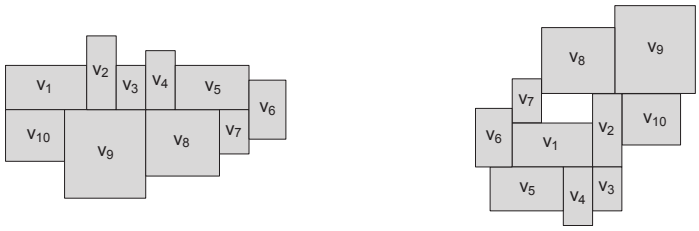
Fig. 3. Star Forest algorithm.

to 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. To solve the Knapsack instances, we use the polynomial-time approximation scheme described in [11].

Finally, the computed solutions for individual stars are packed together is 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. 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}.



(a) Realizing a cycle (b) Realizing a path

Fig. 4. CYCLE COVER algorithm.

Although the heaviest cycle cover can be found in polynomial time in the size of the graph (see Chapter 10 in [13]), 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. The matching induces a set of vertex-disjoint paths and cycles in G as every 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 $\sum_{i \leq t} w_i < \sum_{i \leq n} w_i/2$, 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).

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 *quantitative* 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 realized. Hence, the metric 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.

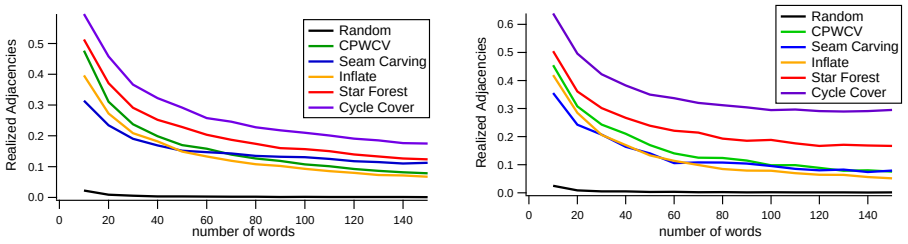
Distortion: This metric is used to measure another aspect of how well the desired similarities are realized, by comparing the distances between all pairs of rectangles to the desired dissimilarities of the words. In order to compute the metric for a given layout, we construct a matrix of ideal distances 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. We modify the definition of distance to reflect the use of non-zero area boxes for vertices (instead of points), by measuring 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: Efficient use of drawing area is a natural goal, captured by this simple metric. We first compute the *used area* as the area covered by words, by adding up the areas of all 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 using the bounding box containing all rectangles, or the area of the convex hull of all rectangles. 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 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,



(a) WIKI dataset, TF-IDF, Cosine Similarity (b) PAPERS, LexRank, Jaccard Similarity

Fig. 5. Realized adjacencies for word clouds of various size.

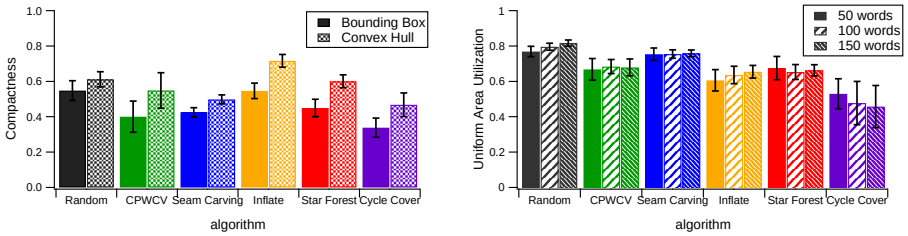
or the amount of uncertainty. To this end, we compute the relative entropy of the variable (induced by a given layout of words) with respect to the uniform distribution [4]: $H = \sum_{i,j} p(i,j) \log \frac{p(i,j)}{q(i,j)}$, where the sum is taken over the created cells, $p(i,j)$ is the actual number 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 distinct words, and (2) PAPERS, a set of 56 research papers published in conferences on experimental algorithms (SEA and ALENEX) in 2011-2012. The texts are preprocessed using the pipeline described in Section 2; that is, for every trial, the words are first ranked and then pairwise similarities between the top $10 \leq n \leq 150$ of them are computed. 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 B). 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. Furthermore, CYCLE COVER performs better than all the existing heuristics for almost all inputs; see Fig. 13(a) in the Appendix B. STAR FOREST also realizes more adjacencies than the existing semantics-aware meth-



(a) WIKI, TF-IDF Ranking, Jaccard Similarity (b) PAPERS, Lex Ranking, Cosine Similarity

Fig. 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.

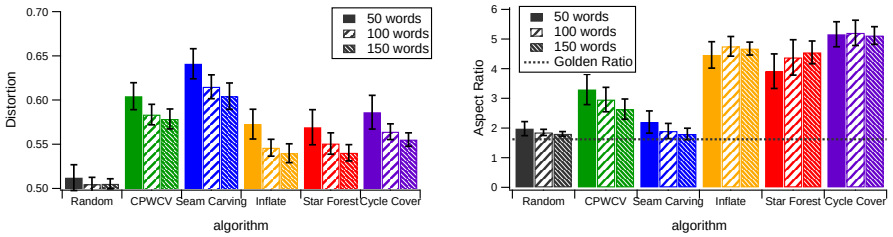
ods. However, for large word clouds (with over 100 words) the difference between STAR FOREST and the existing algorithms drops to $2-5\%$.

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 [21]. Not surprisingly, the RANDOM algorithm realizes only a small fraction of the total weight as it is not designed for preserving semantics.

Compactness: We measure compactness using the bounding box and convex hull of the layout; 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 have similar performance. 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, 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).

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



(a) WIKI, TF Ranking, Cosine Similarity (b) PAPERS, TF Ranking, Cosine Similarity

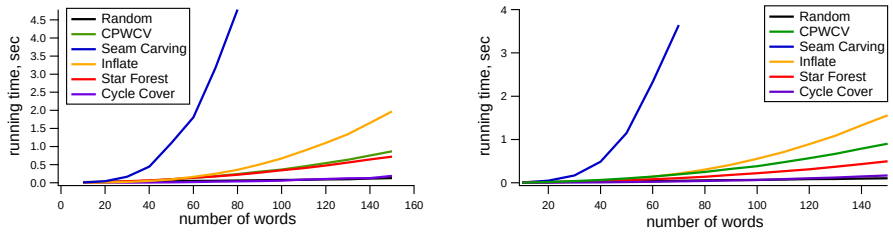
Fig. 7. (a) Measurements of *Distortion*. Note that the y-axis starts at value 0.5. (b) *Aspect Ratio*

compaction step worsens *Distortion* significantly; see Fig. 15. Hence, there is an inherent tradeoff between compactness and 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). INFLATE, STAR FOREST, and CYCLE COVER generate drawings with the aspect ratio from $7:2$ to $9:2$, and the measurements are mostly independent of the number of words in a word cloud. We emphasize here that none of the considered algorithms 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.

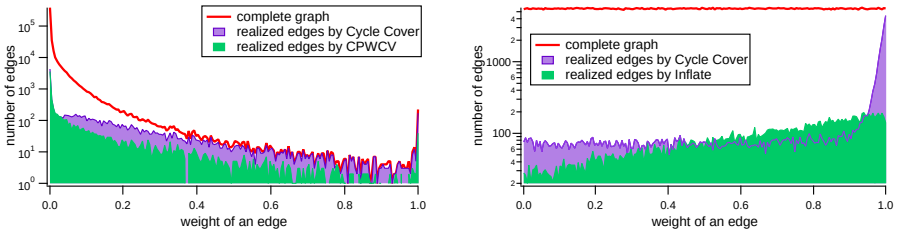
The running times of all but one algorithm are reasonable, dealing with $100-150$ words within 2 seconds; see Fig. 8. The exception is SEAM CARVING, which requires $15-25$ seconds per graph on the PAPERS dataset and $30-50$ on the WIKI dataset.

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 PAPERS datasets) carry half of the total edge weight. It is known that words in text documents follow this distribution (Chapter 5 in [14]) 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



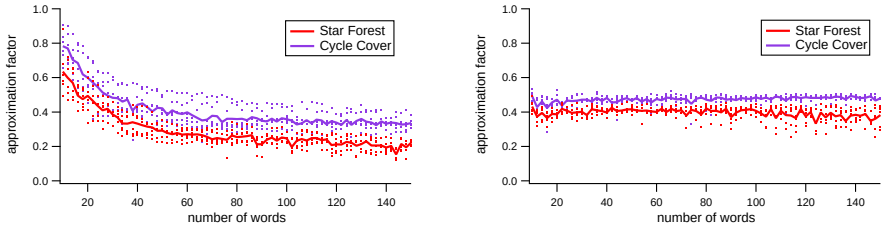
(a) WIKI, TF Ranking, Cosine Similarity (b) PAPERS, TF-IDF, Jaccard Similarity

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



(a) (b)

Fig. 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 (PAPERS dataset). (b) A complete graph in which edge weights are randomly chosen between 0 and 1.



(a) PAPERS, TF Ranking, Cosine Similarity (b)

Fig. 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.

as it realizes 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(b).

None of the presented semantics-preserving algorithms make any guarantees about the optimality of realized edge weights when the input graph is complete (as in real-world examples). 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 [7], 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. See also Appendix A for more details on approximation guarantees.

6 Conclusions and Future Work

We quantitatively evaluated six semantic word cloud methods. 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.

References

1. L. Barth, S. I. Fabrikant, S. Kobourov, A. Lubiw, M. Nöllenburg, Y. Okamoto, S. Pupyrev, C. Squarcella, T. Ueckerdt, and A. Wolff. Semantic word cloud representations: Hardness and approximation algorithms. In *LATIN*, LNCS. Springer, 2014.
2. M. Bekos, T. Dijk, M. Fink, P. Kindermann, S. Kobourov, S. Pupyrev, J. Spoerhase, and A. Wolff. Improved approximation algorithms for semantic word clouds. Technical report, U. of Arizona. `ftp://ftp.cs.arizona.edu/reports/2014/TR14-01.pdf`.
3. A. L. Buchsbaum, E. R. Gansner, C. M. Procopiuc, and S. Venkatasubramanian. Rectangular layouts and contact graphs. *ACM Transactions on Algorithms*, 4(1), 2008.
4. T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley-Interscience, 2006.
5. W. Cui, Y. Wu, S. Liu, F. Wei, M. Zhou, and H. Qu. Context-preserving dynamic word cloud visualization. *IEEE Computer Graphics and Applications*, 30(6):42–53, 2010.
6. S. Deutsch, J. Schrammel, and M. Tscheligi. Comparing different layouts of tag clouds: Findings on visual perception. In *Human Aspects of Visualization*, pages 23–37, 2011.
7. 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.
8. G. Erkan and D. R. Radev. LexRank: graph-based lexical centrality as salience in text summarization. *Journal of Artificial Intelligence Research*, 22(1):457–479, 2004.
9. S. Felsner. Rectangle and square representations of planar graphs. In *Thirty Essays on Geometric Graph Theory*, pages 213–248. Springer, 2013.
10. K. Koh, B. Lee, B. H. Kim, and J. Seo. Maniwordle: Providing flexible control over Wordle. *IEEE Trans. Vis. Comput. Graphics*, 16(6):1190–1197, 2010.
11. E. L. Lawler. Fast approximation algorithms for knapsack problems. *Mathematics of Operations Research*, 4(4):339–356, 1979.
12. H. Li and N. Abe. Word clustering and disambiguation based on co-occurrence data. In *Int. Conf. Comput. Linguistics*, volume 2, pages 749–755, 1998.
13. L. Lovász and M. Plummer. *Matching Theory*. Akadémiai Kiadó, Budapest, 1986.
14. C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.
15. A. Nocaj and U. Brandes. Organizing search results with a reference map. *IEEE Trans. Vis. Comput. Graphics*, 18(12):2546–2555, 2012.
16. M. F. Porter. An algorithm for suffix stripping. *Program: Electronic Library & Information Systems*, 40(3):211–218, 1980.
17. J. Schrammel, M. Leitner, and M. Tscheligi. Semantically structured tag clouds: An empirical evaluation of clustered presentation approaches. In *SIGCHI Conference on Human Factors in Computing Systems*, pages 2037–2040, 2009.
18. F. B. Viégas, M. Wattenberg, and J. Feinberg. Participatory visualization with Wordle. *IEEE Trans. Vis. Comput. Graphics*, 15(6):1137–1144, 2009.
19. F. B. Viégas, M. Wattenberg, F. Van Ham, J. Kriss, and M. McKeon. ManyEyes: a site for visualization at internet scale. *IEEE Trans. Vis. Comput. Graphics*, 13(6):1121–1128, 2007.
20. J. Wang. Clustered layout word cloud for user generated online reviews. Master's thesis, Virginia Polytechnic Institute and State University, 2012.
21. Y. Wu, T. Provan, F. Wei, S. Liu, and K.-L. Ma. Semantic-preserving word clouds by seam carving. *Computer Graphics Forum*, 30(3):741–750, 2011.

Appendix A

Here we provide more details on the formal model on computing semantics-preserving word clouds introduced in [1]. The problem is related to contact representations of graphs and was also considered recently in [2].

The model is based on a weighted graph as described in Section 1. The vertices in the graph (called the *supporting graph*) are the words, and the weights of edges correspond to the strength of the relation between the words. The goal is to represent the supporting graphs by contact of axis-aligned boxes. Not every graph can be represented by touching boxes with fixed dimensions. Moreover, it is NP-hard to decide if there exists a representation of the input graph with the given boxes [1]. The problem remains strongly NP-hard even if restricted to trees and is weakly NP-hard if restricted to stars. This leads to an optimization version of the problem.

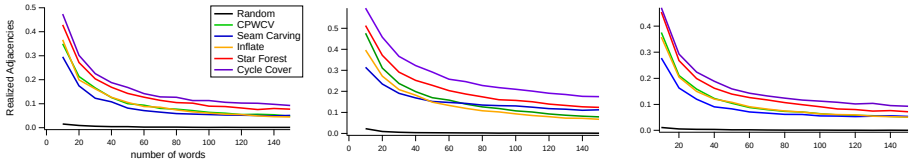
For the optimization version, several algorithms for certain classes of graphs exist [1]. If the supporting graph is a matching or a path, then it can always be realized by contacts of boxes, which are simply arranged along a horizontal line. A similar idea can be applied for a cycle, which can always be realized. If the supporting graph is a star, then there is a constant-factor approximation algorithm based on the GENERALIZED ASSIGNMENT PROBLEM (GAP, a generalization of MULTIPLE KNAPSACK). Using an $\alpha = 1 - 1/e \approx 0.632$-approximation algorithm for GAP, we get an α-approximation for the word cloud problem on a star.

Using the exact algorithm for cycles and the approximation algorithm for stars as building blocks, we devise approximation algorithms for several classes of graphs utilizing the following idea. We partition the edges of the supporting graph G into k subgraphs, apply an α-approximation algorithm for each of the subgraphs, and take the best of the k solutions. The method yields an α/k-approximation algorithm for G. Indeed, consider an optimum solution. By the pigeon-hole principle, one of the subgraphs contains at least $1/k$ of the edges realized by contacts in the optimum solution. Hence, our algorithm produces a solution with weight at least α/k of the optimum. We apply the idea for trees and planar graphs. A tree can be partitioned into 2 star forests (disjoint union of stars) and a planar graph can be partitioned into 5 star forests. Applying the method for stars for each of the forests and combining solutions together, yields the algorithm with approximation factors $\alpha/2$ and $\alpha/5$. Further, any graph of maximum degree 2Δ can be partitioned into Δ sets of edge-disjoint cycles and paths. Therefore, the partitioning results in an $(1/\Delta)$-approximation algorithm for the word cloud problem on graphs of bounded maximum degree.

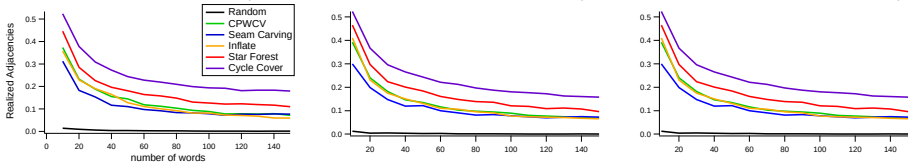
Very recently, we combined the approaches for cycles and stars to produce an algorithm for general supporting graphs with arbitrary edge weights [2]. The algorithm guarantees approximation ratio $40\alpha/3 \approx 21.1$ for *Realized Adjacencies*. We stress that the algorithm involves approximating a number of GAP instances as a subroutine, which relies on solving linear programs. Firstly, the approach is relatively slow for real-world documents. Secondly, the resulting solutions might worsen measurements for the other metrics (for example, *Aspect Ratio*). The heuristics STAR FOREST and CYCLE COVER are designed so as to address the issues. Our experiments indicate that both new algorithms produce results which are only $3 - 5$ times less than the optimum; see Fig. 10.

Appendix B

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.

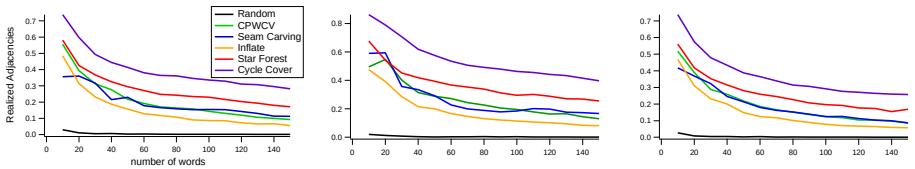


TF, Cosine Similarity | TF-IDF, Cosine Similarity | LexRank, Cosine Similarity

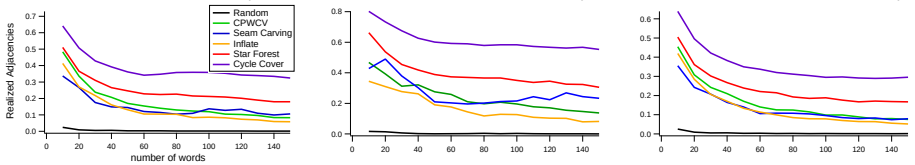


TF, Jaccard Similarity | TF-IDF, Jaccard Similarity | LexRank, Jaccard Similarity

Fig. 11. Realized adjacencies for the WIKI dataset.



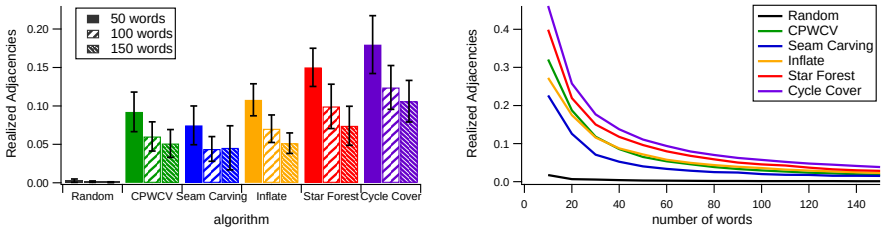
TF, Cosine Similarity | TF-IDF, Cosine Similarity | LexRank, Cosine Similarity



TF, Jaccard Similarity | TF-IDF, Jaccard Similarity | LexRank, Jaccard Similarity

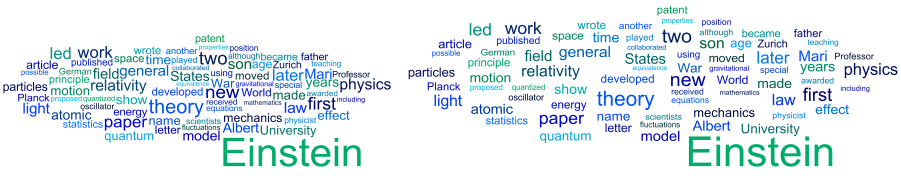
Fig. 12. Realized adjacencies for the PAPERS dataset.

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, Mike McCambridge, Martin Nöllenburg, Yoshio Okamoto, Shiv Pande, Günter Rote, Claudio Squarcella, Alexander Wolff, and Ben Yee, for productive discussions about this problem. Further acknowledgements go to Jixiang Li, Jiankun Lu, and Zixiang Zhou for their help with the implementation of the semantic word cloud online tool, `http://wordcloud.cs.arizona.edu`.



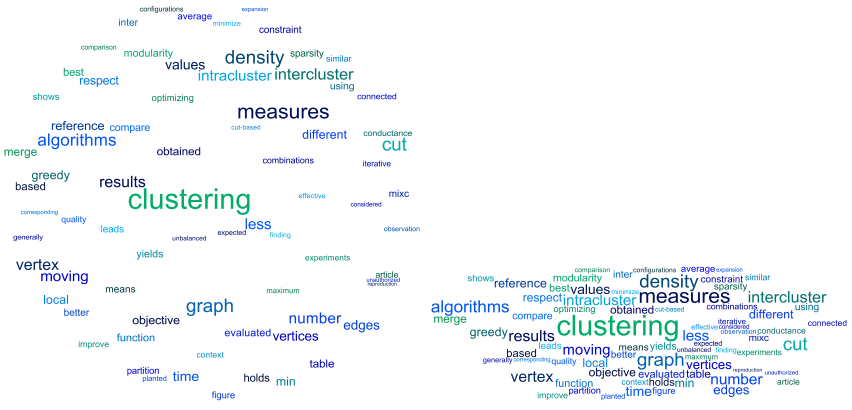
(a) WIKI, TF Ranking, Cosine Similarity (b)

Fig. 13. (a) Standard deviation of realized adjacencies. (b) Realized adjacencies for a complete graph in which edge weights are randomly chosen between 0 and 1.



(a) (b)

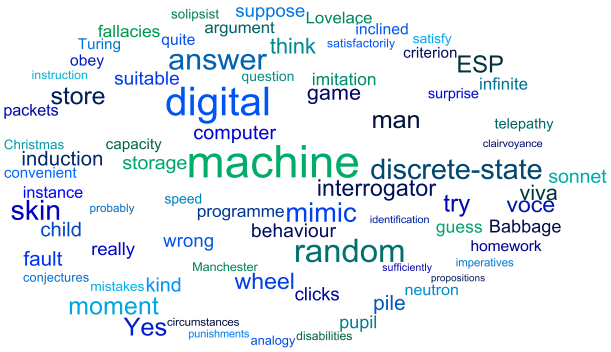
Fig. 14. Wikipedia article "Albert Einstein". (a) Overcompacted word cloud constructed with the INFLATE algorithm. (b) Result after increasing rectangles by 20%.



(a) (b)

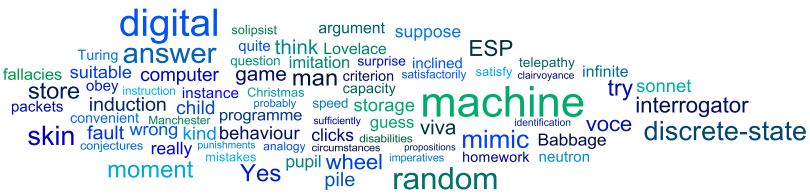
Fig. 15. ALENEX paper "Experiments on Density-Constrained Graph Clustering" by Görke *et al.* (a) Initial layout found by MDS with *Distortion* $\beta = 0.68$. (b) Final result constructed by CPWCV with worsened value of $\beta = 0.57$.

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



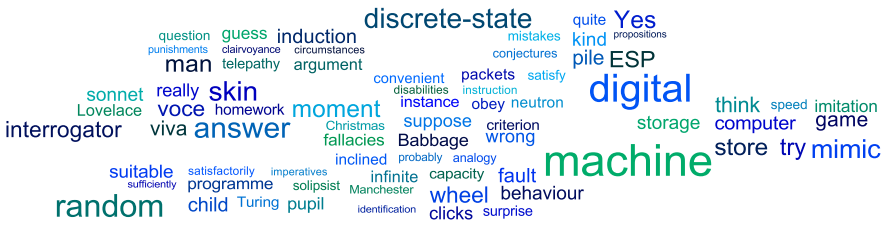
(b) SEAM CARVING: $\alpha = 0.12$, $\beta = 0.59$, $\gamma = 0.47$, $\delta = 0.79$

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

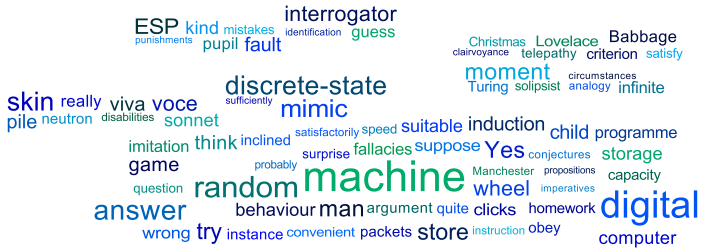


(d) INFLATE: $\alpha = 0.15$, $\beta = 0.54$, $\gamma = 0.75$, $\delta = 0.72$

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



(a) STAR FOREST: $\alpha = 0.23$, $\beta = 0.53$, $\gamma = 0.56$, $\delta = 0.7$



(b) CYCLE COVER: $\alpha = 0.35$, $\beta = 0.56$, $\gamma = 0.61$, $\delta = 0.61$

Fig. 17. "Computing Machinery and Intelligence" by A. M. Turing (top 75 words).

Fig. 18. Der Koalitionsvertrag im Schnellcheck (Quick overview of the coalition agreement), Der Spiegel, 27/11/2013.