Generating Acrostics via Paraphrasing and Heuristic Search

Benno Stein **Matthias Hagen** **Christof Bräutigam**
Bauhaus-Universität Weimar, Germany
`<first name>.<last name>@uni-weimar.de`

Abstract

We consider the problem of automatically paraphrasing a text in order to find an equivalent text that contains a given acrostic. A text contains an acrostic, if the first letters of a range of consecutive lines form a word or phrase. Our approach turns this paraphrasing task into an optimization problem: we use various existing and also new paraphrasing techniques as operators applicable to intermediate versions of a text (e.g., replacing synonyms), and we search for an operator sequence with minimum text quality loss. The experiments show that many acrostics based on common English words can be generated in less than a minute. However, we see our main contribution in the presented technology paradigm: a novel and promising combination of methods from Natural Language Processing and Artificial Intelligence. The approach naturally generalizes to related paraphrasing tasks such as shortening or simplifying a given text.

1 Introduction

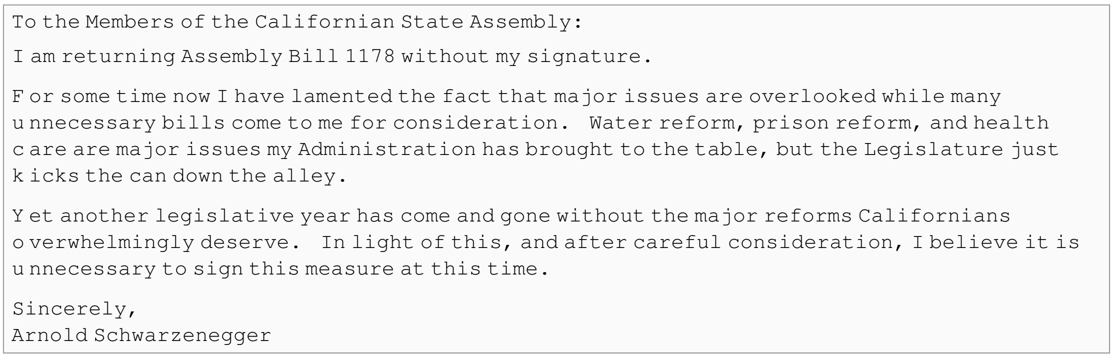
Given some text, paraphrasing means to rewrite it in order to improve readability or to achieve other desirable properties while preserving the original meaning (Androutsopoulos and Malakasiotis, 2010). The paper in hand focuses on a specific paraphrasing problem: rewriting a given text such that it encodes a given acrostic. A text contains an acrostic if the first letters of a range of consecutive lines form a word or phrase read from top to bottom. A prominent and very explicit example of former Governor Schwarzenegger is shown in Figure 1 (see the third and fourth paragraphs). Schwarzenegger himself characterized the appearance of that acrostic a "wild coincidence".[1] However, such a coincidence is highly unlikely: Using the simplistic assumption that first letters of words are independent of each other if more than ten words are in between (line length in the Schwarzenegger letter) and calculating with the relative frequencies of first letters in the British National Corpus (Aston and Burnard, 1998), the probability for the acrostic in Figure 1 can be estimated at $1.15 \cdot 10^{-12}$. Typically, a given text will not contain a given acrostic but has to be reformulated using different wording or formatting to achieve the desired effect. Thus we consider the purposeful generation of acrostics a challenging benchmark problem for paraphrasing technology, which is subject to soft and hard constraints of common language usage.

The paper shows how heuristic search techniques are applied to solve the problem. Different paraphrasing techniques are modeled as operators applicable to paraphrased versions of a text. By pruning the so-formed search space and by employing a huge corpus of text n-grams for the possible operators, we are able to generate acrostics in given texts. Our algorithmic solution is a novel combination of techniques from Natural Language Processing and Artificial Intelligence. We consider such combinations as a very promising research direction (Stein and Curatolo, 2006; Sturtevant et al., 2012), and we point out that the problem of acrostic generation serves as a serious demonstration object: the presented model along with the heuristic search approach generalizes easily to other paraphrasing tasks such as text shortening, improving readability, or e-journalism.

2 Related Work and Problem Definition

Rewriting a given text in order to "encode" an acrostic is a paraphrasing problem, which in turn is studied in the domain of computational linguistics and natural language processing. We review relevant literature

[1] `www.huffingtonpost.com/2009/10/30/schwarzenegger-f-bomb-in_n_340579.html`, last accessed: June 12, 2014.



```
To the Members of the Californian State Assembly:
I am returning Assembly Bill 1178 without my signature.

For some time now I have lamented the fact that major issues are overlooked while many
unnecessary bills come to me for consideration.  Water reform, prison reform, and health
care are major issues my Administration has brought to the table, but the Legislature just
kicks the can down the alley.

Yet another legislative year has come and gone without the major reforms Californians
overwhelmingly deserve.  In light of this, and after careful consideration, I believe it is
unnecessary to sign this measure at this time.

Sincerely,
Arnold Schwarzenegger
```

Figure 1: Excerpt from a letter of former Governor Arnold Schwarzenegger to the Californian State Assembly in October 2009. The third and fourth paragraphs contain the acrostic “F$***$ You”.

of the topic and highlight techniques that will be employed in our work.

An important branch of the paraphrasing literature focuses on analyses with fixed corpora. Such corpora typically are *parallel* in the sense that they contain different formulations of the same facts (Barzilay and McKeown, 2001; Barzilay and Lee, 2003; Callison-Burch, 2008). These “facts” can be news articles on the same event (Clough et al., 2002; Dolan and Brockett, 2005), different translations of a source text to a target language (Pang et al., 2003), or cross-lingual parallel corpora (Bannard and Callison-Burch, 2005). As most of the early parallel corpora were constructed manually (especially the judgments of whether a pair of sentences forms a paraphrase), there are two shortcomings. First, the obtained paraphrases are usually specific to the domain covered in the corpus (e.g., showbiz news) and often do not generalize well. Second and probably more severe is the fact that manually building parallel corpora is very expensive, such that the available ones are rather small: the METER corpus contains only 1717 texts on legal issues and showbiz (Clough et al., 2002), the MSRP corpus contains only 5801 sentence pairs (Dolan and Brockett, 2005). Recently, new methods employ machine learning techniques to automatically build larger paraphrase collections from parallel corpora (Ganitkevitch et al., 2011; Ganitkevitch et al., 2013; Metzler et al., 2011; Metzler and Hovy, 2011). We include the paraphrase database (Ganitkevitch et al., 2013)—a database of extracted patterns from such large scale corpora—as one source of potential paraphrases in our algorithm.

Compared to the large body of literature that “extracts” paraphrases from (parallel) corpora, there is relatively little work on automatically paraphrasing a given text. Some of the early generation methods are based on rules that encode situations wherein a reformulation is possible (Barzilay and Lee, 2003). A problem with rules is that often the rather complicated patterns extracted from text corpora are hardly applicable to a given to-be-paraphrased text: manually created corpora are simply too small and machine generated paraphrasing rules often do not match in a given text. Other early methods use machine translation (Quirk et al., 2004). However, the need for large and expensive parallel manual translation corpora cannot be circumvented by using multiple resources (Zhao et al., 2008).

Another branch of paraphrasing methods is based on large thesaurus resources such as WordNet (Fellbaum, 1998). The idea is to insert synonyms into a text when the context fits (Bolshakov and Gelbukh, 2004; Kauchak and Barzilay, 2006). The most recent approaches are statistics-based (Chevelu et al., 2009; Chevelu et al., 2010; Zhao et al., 2009; Burrows et al., 2013).

Compared to the existing research, we have a more difficult use case here. Existing paraphrase generation focuses on sentence paraphrasing, while we have to consider a complete text that has to be rewritten/reformatted in order to contain a given acrostic. We will employ the above shown state-of-the-art paraphrasing procedures as ”operators” in our approach. This new problem setting of applying different operators to a complete text forms a search problem with a huge search space. In order to deal with this search space, we apply powerful search heuristics from Artificial Intelligence. The combination of heuristic search with established text level paraphrasing techniques represents a new approach to tackle problems in computational linguistics. The acrostic generation problem is defined as follows:

ACROSTIC GENERATION

Given: (1) A text T and an acrostic x.
(2) A lower bound $l_{\min}$ and an upper bound $l_{\max}$ on the desired line length.

Task: Find a paraphrased version T^* of T in monospaced font that encodes x in some of its lines when possible. Each line of T^* has to meet the length constraints.

3 Modeling Paraphrasing as Search Problem

This section shows how to model paraphrasing in general and ACROSTIC GENERATION in particular as a search problem (Pearl, 1984). The search space is a universe $\mathcal{T}$ of candidate texts for which we can devise, at least theoretically, a systematic search strategy: if $\mathcal{T}$ is finite, each element $n \in \mathcal{T}$ is analyzed exactly once. The elements in $\mathcal{T}$ represent states (nodes), and there is a limited and a-priori known set of possibilities (edges, paraphrasing operators) to get from a node n to an adjacent or successor node n_i. A paraphrasing operator ϕ provides a number of parameters that control its application. Each state $n \in \mathcal{T}$ is considered an acrostic (sub)problem; the dedicated state $s \in \mathcal{T}$ represents the original problem while $\Gamma \subset \mathcal{T}$ is the set of solution nodes that have no problem associated with. The following subsections will outline important properties of the search space and introduce a suited cost measure to control the exploration of $\mathcal{T}$.

3.1 Search Space Structure

Solving an instance of ACROSTIC GENERATION under a so-called state-space representation means to find a *path* from s, which represents the original text T, to some goal state $\gamma \in \Gamma$. The problem of finding an acrostic consists of tightly connected subproblems (finding subsequences of the acrostic) that cannot be solved independently of each other. Most puzzles such as Rubik's cube are of this nature: changing a decision somewhere on the solution path will affect all subsequent decisions. By contrast, a so-called problem-reduction representation will exploit the fact that subproblems can be solved independently of each other. Many tasks of logical reasoning and theorem proving give rise to such a structure: given a set of axioms, the lemmas required for a proof can be derived independently, which in turn means that the sought solution (a plan or proof) is ideally represented by a *tree*.

Searching $\mathcal{T}$ under a state-space representation means to unfold a tree whose inner nodes link to successor nodes that encode alternative decisions; hence these inner nodes are also called OR-nodes. Each path from s that can be extended towards a goal state γ forms a *solution candidate*. Similarly, searching $\mathcal{T}$ under a problem-reduction representation also means to unfold a tree—however, the tree's inner nodes must be distinguished as AND-nodes and OR-nodes, whereas the successors of an AND-node encode subproblems all of which have to be solved. A solution candidate then is a tree comprised of (1) the root s, (2) OR-nodes with a single successor, and (3) AND-nodes with as many successors as subproblems all of which are characterized by the fact of being extensible towards goal states in Γ. Figure 2 contrasts both search space structures.

Under either representation, OR-graphs as well as AND-OR-graphs, the application of a sequence of

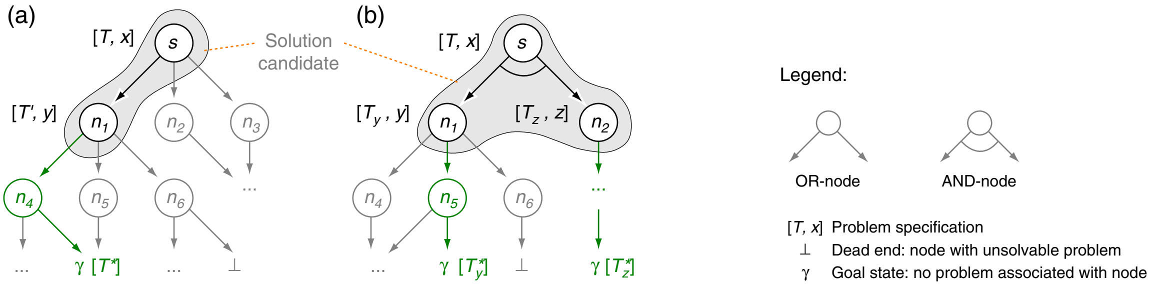


Figure 2: (a) State-space representation (OR-graph) versus (b) Problem-reduction representation (AND-OR-graph). OR-nodes encode alternative decisions, while AND-nodes decompose a problem into subproblems all of which are to be solved.

operators will easily lead to situations where states are revisited—precisely: are generated again. Search algorithms maintain so-called OPEN- and CLOSED-lists to manage the exploration status of generated nodes. However, because of the intricate state encoding, which must inform about the effect of all applied operators from s to an arbitrary node, the exponentially growing number of nodes during search, and the necessity of efficiently querying these lists, sophisticated data structures such as externalized hash tables and key value stores are employed. Typically, these data structures are tailored to the problem domain (here: to paraphrasing), and they model heuristic access strategies to operationalize a probabilistically controlled trade-off between false positive and true negative answers to state queries.

We have outlined the different search space structures since ACROSTIC GENERATION may show an OR-graph puzzle nature at first sight: paraphrasing at some position will affect all following text. Interestingly, there is a limited possibility to introduce "barriers" in the text, which allows for an AND-OR-graph modeling and hence for an isolated subproblem treatment. Examples include paraphrasing operators that do not affect line breaks, or acrostics consisting of several words and thus spanning several paragraphs.

Since in general the underlying linguistic considerations for the construction and maintenance of such barriers are highly intricate and complex, we capture this structural constraint probabilistically as shown in Equation (1). The equation models the problem difficulty or *effort for acrostic generation*, E, and introduces $P_{y \circ z}(|x|)$, which quantifies the probability for the event that an acrostic x can be treated independently as two partial acrostics y and z, where $x = yz$.

$$E(x) = \begin{cases} e(x) & \text{If } |x| = 1. \\ P_{y \circ z}(|x|) \cdot \Big(E(y) + E(z)\Big) + (1 - P_{y \circ z}(|x|)) \cdot E(y) \cdot E(z) & \text{If } |x| > 1. \end{cases} \tag{1}$$

Remarks. The effort for generating a single-letter acrostic of length 1 is $e(x)$, with $e(x) \geq 1$. Based on $e(x)$, we recursively model the *true effort* $E(x)$ for generating an acrostic $x = yz$ as follows: as additive effort if the generation of the acrostics y and z can be done independently, and as multiplicative effort otherwise. Observe how $P_{y \circ z}$ controls the search space structure: if $P_{y \circ z} = 0$ for all partitionings of x into y and z, one obtains a pure state-space representation for ACROSTIC GENERATION. Similarly, the other extreme with $P_{y \circ z} = 1$ results in a series of $|x|$ letter generation problems that can be solved independently of each other.

As an estimate $\hat{e}(x)$ for $e(x)$ we suggest the multiplicative inverse of the occurrence probabilities of the first letters in the English language, as computed from the British National Corpus (BNC). The BNC is a 100 million word collection of written and spoken language from a wide range of sources, designed to represent a wide cross-section of current British English (Aston and Burnard, 1998). The BNC probabilities vary between 0.115719 for the letter "s" and 0.00005 for the letter "z". As an estimate for $P_{y \circ z}$ we suggest the $N(5, 0.5)$ distribution to model the paragraph lengths in T or, equivalently, the number of characters of the (English) words in x. These choices give rise to $\hat{E}(x)$, the *estimated effort* for generating an acrostic x. $\hat{E}(x)$ is used to assess the (residual) problem complexity and, under a maximum likelihood approach, models the expected search effort. There is a close relation between the effort estimate $\hat{E}$ and the quality estimate $\hat{Q}$ introduced below, which will be exploited later on, in Equation (3).

3.2 Cost Measure Structure

Cost measures—equivalently: merit measures—form the heart of systematic search strategies and determine whether an acceptable solution of a complex problem can be heuristically constructed within reasonable time. Here, we refer to general best-first search strategies as well as variants that relax strict admissibility. As a working example consider the following text T about Alan Turing taken from the English Wikipedia, where the task is to generate the acrostic $x =$ `Turing` with $l_{\text{min}} = 55$ and $l_{\text{max}} = 60$.

```
Alan Mathison Turing was a British mathematician, logician,
cryptanalyst and computer scientist. He was highly influential in
the development of computer science, giving a formalization of the
concepts of algorithm and computation with the Turing machine,
which can be considered a model of a general purpose computer.
```

A possible solution T^* (a paragraph's last line may be shorter than $l_{\min}$) :

```
The British mathematician Alan Mathison Turing was also an
unrivaled logician, cryptanalyst and computer scientist. He
revolutionized the development of computer science, giv-
ing a formalization of the concepts of algorithm and defi-
nite computation with the Turing machine, which can be re-
garded a model of a general purpose computer.
```

T^* is of a high quality though it introduces an exaggerating tone, this way violating Wikipedia's neutrality standard. Also note that the applied paraphrasing operators vary in their quality, which is rooted in both the kind and the context of the operators. Table 1 (left) shows a selection of the operators, some of which are applied in a combined fashion. Section 4 introduces the operators in greater detail.

To further formalize the quantification of a cost measure C or a merit measure Q, we stipulate on the following properties:

1. The quality of the original text T cannot be improved. Each paraphrasing operator ϕ introduces unavoidable deficiencies in T.
2. The overall quality of a solution T^* depends on the quality of all applied paraphrasing operators.
3. Following readability theory and relevant research, the severity of text deficiencies—here introduced by a paraphrasing operator ϕ—has a disproportionate impact on the text quality (Meyer, 2003).
4. To render different problems and solutions comparable, the achieved quality of a solution T^* has to be normalized.

Equation (2) below shows the basic structure of Q, the proposed, unnormalized merit measure. Its optimization yields Q^*. $Q^*(n)$ assigns to a node $n \in \mathcal{T}$ the maximum paraphrasing quality of a text T^* that contains the partial acrostic associated with n. Likewise, $Q^*(s)$ characterizes the quality of the optimum solution for solving ACROSTIC GENERATION.

$$\frac{1}{Q^*(n)} = \begin{cases} 0 & \text{If } n \in \Gamma. \\ \min\limits_{i} \left\{ \dfrac{1}{q(n, n_i)} + \dfrac{1}{Q^*(n_i)} \right\} & \text{Otherwise.} \end{cases} \tag{2}$$

Remarks. The state (node) n_i denotes a direct successor of the state (node) n in the search space $\mathcal{T}$. Associated with n_i is a text resulting from the application of a paraphrasing operator ϕ to the text associated with n, whereas $q(n, n_i)$ quantifies the *local quality* achieved with ϕ. The measure in Equation (2) is both of an additive form and formulated as a minimization problem. As shown in the following, it can be reformulated for a best-first algorithm scheme, ensuring admissibility under a delayed termination condition. Also note that the merit measure operationalizes the above Property 3 via the harmonic mean computation. Accordingly, we obtain a normalized *overall quality* $\bar{Q}^*$ given an acrostic x as $\bar{Q}^* = |x| \cdot Q^*$.

To turn Equation (2) into actionable knowledge, the quality $q(n, n_i)$ of a paraphrasing operator ϕ when moving from n to a successor n_i needs to be quantified. We employ for q the domain $[0; 1]$, where 0 and 1 encode the worst and best achievable quality respectively. By construction the normalized quality $\bar{Q}^*$ will then lie in the interval $[0; 1]$ as well, thus greatly simplifying the interpretation of the measure.

Table 1 (right) shows values for the local quality of the operators in the Alan Turing example, which are derived from linguistic quality considerations and the experimental analysis detailed in Section 5. The comment column argues the linguistic meaningfulness. If we agree on $q = 1.0$ for the first two lines of the generated acrostic $x =$ `Turing` and recursively apply the merit measure defined in Equation (2), we obtain $Q = 0.127$ as unnormalized and $\bar{Q} = |x| \cdot Q = 0.76$ as normalized overall quality.

To make Equation (2) applicable as cost estimation heuristic $f(n)$ in a best-first algorithm scheme, Equation (3) below unravels its recursive structure in the usual way as $f(n) = g(n) + h(n)$. The semantics is as follows: under an optimistic estimate $h(n)$ (= underestimating costs or overestimating merits) the

Table 1: Left: Paraphrasing operators in the Alan Turing example. Right: Values for the local quality of the respective operators, which entail the normalized overall quality $\hat{Q} = 0.76$ for the example.

Line	Operator ϕ	Text → paraphrased text	$q(n, n_i)$	Comment
3	synonym	highly influential → **r**evolutionized	0.9	stylistically well, exaggerating tone
4	hyphenation	giving → giv-**i**ng	0.6	unexpected hyphen for a short word
5	tautology	computation → defi-**n**ite computation	0.6	tautology arguable, hyphen unusual
6	synonym	considered → re-**g**arded	0.7	synonym suited, hyphen acceptable

total cost (the overall quality) for solving ACROSTIC GENERATION via a path *along node* n is always larger (smaller) than $f(n)$. In particular, $g(n)$ accumulates the true cost (the achieved quality) for the partial acrostic via a concrete path $s = n_0, n_1, \ldots, n_k = n$, while $h(n)$ gives an underestimation of the cost (overestimation of the quality) for the remaining part of the acrostic. Observe that the additive form of Equation (2) guarantees the parent discarding property (Pearl, 1984), which states that no decision on a path from n to a goal state γ can change the value for $g(n)$.

A tricky part is the construction of $h(n)$, which, on the one hand, may ensure admissibility, while, on the other hand, should be as close as possible to the real cost. Here, the measure $E(x)$ for the problem difficulty from Equation (1) comes into play, which models the problem decomposability and which informs us about the largest remaining subproblem (= the depth of the deepest remaining OR-graph) when solving x. Without loss of generality, admissibility is ensured if (a) the probability $P_{y \circ z}$ used in $\hat{E}(x)$ is biased towards decomposability, and if (b) we assume that the remaining acrostic x can be solved by always applying the cheapest (maximum quality-preserving) operator $q_{\max}$.

$$\underbrace{\frac{1}{\hat{Q}(n)}}_{f(n)} = \underbrace{\sum_{i=1}^{k} \frac{1}{q(n_{i-1}, n_i)}}_{g(n)} + \underbrace{\log_K \left(\hat{E}(\tau(n))\right) \cdot \frac{1}{q_{\max}}}_{h(n)}, \quad \text{where } n_0 = s,\ n_k = n \tag{3}$$

Remarks. $\tau(n)$ denotes the remaining acrostic x that is associated with node $n \in \mathcal{T}$. The logarithm base K serves for normalization purposes with regard to the BNC letter frequencies $\hat{e}(x), |x| = 1$, which are used within $\hat{E}(x)$ in Equation (1). We define K as the multiplicative inverse of the occurrence probability of the least frequent letter in the remaining acrostic $x = \tau(n)$, which gives rise to the inequality $\log_K(\hat{E}(x)) \leq |x|$. This choice entails two properties: (1) it underestimates the remaining acrostic length and hence ensures the admissibility characteristic of $h(n)$, and, (2) it yields an increasing accuracy of $h(n)$ when approaching a goal state in Γ. Finally, we can substitute 1.0 as an upper bound for $q_{\max}$, again preserving the admissibility of $h(n)$.

Admissibility, i.e., the guarantee of optimality during best-first search, may not be the ultimate goal: if $h(n)$ underestimates costs (overestimates merits) too rigorously, best-first search degenerates to a kind of breadth-first search—precisely: to uniform-cost search. Especially if computing power is a scarce resource, we may be better off with a depth-preferring strategy. Observe that the logarithm base K in Equation (3) provides us a means to smoothly vary between the two extremes, namely by choosing K from $[K_{\min}; K_{\max}]$, where $K_{\min}$ ($K_{\max}$) specifies the multiplicative inverse of the occurrence probability of the most (least) frequent letter in the remaining acrostic $x = \tau(n)$.

4 Paraphrasing Operators

Most of the following operators used in our heuristic search process employ state-of-the-art linguistic tools or are based on standard knowledge from Wikipedia. Table 2 shows information about the role of individual operators in our experiments from Section 5; the table illustrates also the effort for preparing (offline) and applying (online) the operators.

4.1 Context-Independent Operators

Line break Since we are dealing with text that should spread over several lines, breaking between lines is one of the most basic operators. Similarly, it is the most efficient operator, and Column 4 of Table 2 illustrates the performance of the others in relation to this operator. Line breaks are possible at the end of sentences (i.e., a paragraph break), while a line break in between words is only possible if it falls in the $[l_{\min}; l_{\max}]$-window given by the line length constraints.

Hyphenation Related to line breaks are hyphenations. We re-implemented and employ the standard TEX hyphenation algorithm (Knuth, 1986). Analogous to line breaks, hyphenation is applicable if the line after hyphenating (and line breaking) has a length in the $[l_{\min}; l_{\max}]$-window.

Function word synonym Specific groups of so-called synsemantic words can often be replaced by each other without changing a text's meaning. We have identified 40 such groups from a list of Sequence Publishing[2] and the Paraphrase Database (Ganitkevitch et al., 2013). Examples are {can, may}, {so, thus, therefore, consequently, as a result}, and {however, nevertheless, yet}.

Contraction and expansion Some local text changes can be achieved by contracting or expanding formulations like "he'll" or "won't". We have identified 240 such pairs from Wikipedia.[3] Other possibilities are to spell out / contract standard abbreviations and acronyms. We have mined a list of several thousand such acronyms from the Web.[4] Finally, also small numbers can be spelled out or be written as numerals (e.g., "five" instead of "5"). It is interesting to note that this operator was hardly ever used on successful paths in our experiments.

Spelling In principle, we want to generate text that is correctly spelled. In certain situations, however, it can nevertheless be beneficial to introduce some slight mistakes in order to change word lengths or to generate letters not present in the correctly spelled text. We employ a list of 3 000 common misspellings mined from Wikipedia[5] (e.g., "accidently" instead of "accidentally"). We also include several standard typos related to computer keyboards (e.g., an "m" is often typed as an "n" and vice versa) as well as phonetic misspellings (e.g., "f" and "ph" often sound similar). Since the quality score of wrong spellings tends to be low, this operator has to be treated with care. Especially at the beginning of words, typos are less common than within words such that we allow typos only within words.

Wrong hyphenation Similar to wrong spellings is the purposeful choice of a wrong hyphenation. As with wrong spellings the quality score is typically low. We thus employ this operator very carefully, avoiding for instance syllables on a new line with just two letters. Analogous to correct hyphenation, the line length has to be in the $[l_{\min}; l_{\max}]$-window to apply wrong hyphenation. Despite its questionable quality, this operator is used pretty often in the experiments since it has a very high probability of "generating" a desired letter.

4.2 Context-Dependent Operators

Synonym For identifying synonyms, WordNet's synsets (Fellbaum, 1998) is used. Since only a small subset of the synset members of a to-be-replaced word w is reasonable in the context around w in T, we check in the Google n-gram corpus (Brants and Franz, 2006) whether the synonym in fact fits in the same context. In this regard the public and highly efficient Netspeak API (Stein et al., 2010; Riehmann et al., 2012) is employed. For example, given "hello world", the most frequent phrase with a synonym for world is "hello earth". The Google n-grams are up to five words long, such that at most four context words can be checked before or after w. Previous studies showed that more context yields higher quality synonyms (Metzler and Hovy, 2011; Pasca and Dienes, 2005), so that we use at least two words before or after w. Higher quality scores are achieved if the context is matched before as well as after w.

[2] sequencepublishing.com/academic.html\#function-words, last accessed: June 12, 2014

[3] en.wikipedia.org/wiki/List_of_English_contractions\#English, last accessed: June 12, 2014, en.wikipedia.org/wiki/English_auxiliaries_and_contractions, last accessed: June 12, 2014

[4] www.acronymfinder.com, last accessed: June 12, 2014

[5] en.wikipedia.org/wiki/Wikipedia:Lists_of_common_misspellings/For_machines, last accessed: June 12, 2014

Table 2: Statistics for the applicability, usage, and effort of single operators. “Application probability” reports whether an operator is applicable at all at some node, “Usage” reports the application probability on a solution path, “Effort” reports the (online) application effort as multiple of the fastest operator (the Line break operator), and “Offline time” reports the preprocessing time in ms per word before the actual search is started. All numbers are profiled within the experiment setup described in Section 5.

Paraphrasing operator	**Application probability** (in %)	**Usage probability** (in %)	**Effort** (multiple of Line break)	**Offline time** (in ms per word)
Line break	16.14	21.13	1.00	0.00
Hyphenation	5.48	9.38	1.22	1.38
Function word synonym	1.43	2.57	1.33	0.01
Contraction and expansion	0.29	0.00	1.93	0.02
Spelling	6.98	1.57	1.18	0.11
Wrong hyphenation	9.53	37.63	1.07	1.38
Synonyms	16.69	2.47	1.66	23.24
Word insertion or deletion	43.46	25.24	2.46	42.75
Average	**12.50**	**12.50**	**1.49**	**14.35**

Word insertion or deletion Similar to the synonym replacement, the insertion or deletion of short phrases is handled. For all positions of the given text, the Google n-grams are checked with Netspeak (see above) for a word w that sufficiently often appears within the context of the text. Similarly, for each word w in the text, it is checked whether there are sufficiently many n-grams without the word but the same surrounding context. In both cases, w is a candidate to be inserted or deleted. Given “hello world”, the most frequently used intermediate word is “cruel”, yielding the phrase “hello cruel world.” Again, as with synonyms, context size and quality are positively correlated. We thus use at least two words as context and favor variants that match more context.

4.3 Further Operator Ideas

In pilot experiments, also the three operator ideas discussed below were analyzed. The ideas show promising results for specific cases, but they easily lead to unexpected text flows due the introduction of odd sentences or names. The operators require future work to better fit them in the given text’s context, and they are not employed within the experiments in Section 5.

Tautology It is often possible to introduce entirely new phrases or sentences in a text, which may confirm a previous sentence or which introduce a (nearly) arbitrary but true statement. We tested a small list, including among others “As a matter of fact this is true.” or “I didn’t know that until now.” However, due to improper context such tautologies may mess up a text significantly.

Sentence beginning The beginning of a sentence can often be modified without changing its meaning. Possibilities include the addition of function words like “in general” or “actually”, but also the addition of a prefix like “⟨someone⟩ said that . . . ” or “⟨time⟩ ⟨someone⟩ said that . . . ” where ⟨someone⟩ is to be replaced by a person’s name or {I, he, she} depending on the full context of the text (e.g., author’s name or gender of name mentioned before). The ⟨time⟩ expression may expand to “yesterday” or “last week”, etc. Especially with the usage of names, a whole bunch of letters can be generated. However, context is more subtle for this operator compared to the usage of Google n-grams.

Full PPDB The paraphrase database (Ganitkevitch et al., 2013) comes in different sizes and quality levels. Many synonymity relations for nouns are already covered by WordNet, and function word replacements are already an operator on their own. Still, the rich variety of the full data set can form a semantically strong operator. However, in our pilot experiments, the full PPDB patterns often decreased text quality unacceptably, such that we refrained to use PPDB as a single operator in our experiments.

5 Experimental Evaluation

Goal of the evaluation is to show that our approach is able to efficiently generate acrostics in different situations. In this regard, we analyze the general success of acrostic generation, the influence of different operators, and effects on the text quality.

5.1 Experiment Setup

To model different "use cases" in which acrostics have to be inserted, we use texts of different genres: newspaper articles, emails, and Wikipedia articles. We sample 50 newspaper articles from the Reuters Corpus Volume 1 (Lewis et al., 2004), 50 emails from the Enron corpus (Klimt and Yang, 2004), and 50 articles from the English Wikipedia. Each text contains at least 150 words excluding tables and lists.

As target acrostics for all of the above text types, the 25 most common adjectives, nouns, prepositions, verbs, and 50 other common English words are chosen (in total 150 words).[6] This scenario reflects the inclusion of arbitrary words. Other target acrostics are formed by the 100 most common male and female first names from the US (in total 200 words).[7] This models the standard poetry usage of acrostics where often a writer's name is encoded. For all input texts, we also model self-referentiality by using a text's first phrases as the target acrostics (in total 150 phrases for which at least the first word has to be generated). In these cases, the first letter of the acrostic is also the first letter of the text—a fact that enables the controlled evaluation of the importance of the producibility of the first letter.

The evaluation system is a standard quad-core PC running Ubuntu 12.04 with 16 GB of RAM. A relevant subset of all operator application possibilities is preprocessed and stored in-memory (e.g., the synonym n-gram frequencies for every word), whereas the preprocessing time (about one minute in total per run) is not counted for the search process. We then conduct an A* search using the preprocessed operator tables and an admissible instance of Equation (3). To safe runtime, we slightly transform the problem setting and require the acrostic to start at the beginning of the given text. Pilot experiments show that a good choice for line lengths is $l_{\min} = 50$ and $l_{\max} = 70$. Note that this is only slightly more flexible than a standard line length between 55 and 65 characters (i.e., about 10-12 words) but eases acrostic generation. The experiments also reveal that a successful run (the acrostic can be generated) usually takes less than 30 seconds for the search part. An unsuccessful run (the acrostic cannot be generated) takes five to ten minutes until its termination caused by the memory constraints for the open list.

5.2 Experiment Discussion

Given our hardware and time restrictions, about 20% of the runs are successful altogether. The producibility of the first letter is critical for the overall success: we observe an almost 90% success rate for the self-referential acrostics compared to the about 20% for all others. Statistics for the successful runs are given in Table 3. As can be seen, our system is able to generate about 90 000 nodes with 550 goal checks per second. This yields reasonable answer times on the test acrostics: the average number of goal checks needed when the acrostic can be generated is below 10 000 (about 20 seconds of runtime). Only very few successful runs took more than 40 seconds; the self-referential acrostics that often are two or three words long form the main exception. Not that surprisingly, shorter acrostics are on average generated faster than longer ones. Interestingly, besides self-referential acrostics, male first names seem to be the most difficult acrostics when taking the required runtime into account. Note in this regard that many of the (longer) female names start with a more common first letter, which can be generated faster.

Since our approach is the first attempt at the problem of acrostic generation, we cannot compare to other systems from the literature. Instead, we compare to a baseline system that can only use line breaking and hyphenation as its operators. This also helps to further examine the effect of the producibility of the first letter. Whenever the acrostic's first letter is not the first letter of the text, the baseline fails right from the start: recall that for our experiments we require the acrostic to start at the text's beginning. For less than 1% of our test cases, the baseline can generate the acrostic. Most of these few cases are self-referential first words. Even if the first letter is already present, usually the second or third one are not producible by

[6] `en.wikipedia.org/wiki/Common_English_words`, last accessed: June 12, 2014.

[7] `www.ssa.gov/OACT/babynames/decades/century.html`, last accessed: June 12, 2014.

Table 3: Experimental results for complete acrostic generations. For each of the acrostic types (Column 1), several thousand runs were conducted for which we report averaged values of the acrostic lengths in letters (Column 2). The columns 3-10 relate to successful generations and report averaged values for the runtime in seconds, size of the explored search tree, generated nodes and goal checks per second, used main memory, and the quality change according to the introduced measures.

Acrostic type	**Length** (in letters)	**Runtime** (total in s)	**Nodes** (total)	**Nodes** (per s)	**Goal checks** (per s)	**Memory** (in MByte)	**Quality-related measures**		
							Δ WFC	**Δ ARI**	**Δ SMOG**
Common English words									
Adjective	4.36	3.25	286 960	88 269	578	270	-0.99	-1.61	-0.91
Noun	4.47	3.40	285 016	83 837	576	277	-0.39	-0.96	-0.50
Preposition	3.44	3.16	280 593	88 853	556	243	-1.59	-2.28	-1.29
Verb	3.59	2.76	251 161	90 898	595	236	-0.95	-1.60	-0.92
Other	3.29	2.41	218 974	90 755	601	206	-1.10	-2.05	-1.11
Common US first names									
Male	6.00	9.32	851 665	91 368	554	649	-0.74	-1.87	-0.93
Female	6.07	7.82	740 418	94 693	546	575	-0.60	-1.77	-0.93
Self-referential									
First words	10.33	36.09	3 164 873	87 690	518	1 985	-0.31	-0.09	0.20
Average	**5.19**	**8.53**	**759 957**	**89 545**	**565**	**372**	**-0.83**	**-1.53**	**-0.80**

the simple operators. Using all operators, our approach is able to produce a self-referential acrostic of more than seven characters in 80% of the cases. On average, acrostics of ten characters are possible for the self-referential cases. This further highlights the importance of the first letter: whenever it is producible, the success ratio is much higher.

To compare the importance of the different operators, we count for the successful generations in the experiments of Table 3, how often operators are used and how long the search paths are. Table 2 contains information on the applicability of the different operators. About 21% of the operator applications are line breaks, another 9% are hyphenations. Interestingly, about 38% of the operator applications are wrong hyphenations despite the low quality of this operator. Even though our heuristic tries to avoid wrong hyphenations, there are a lot of situations where all other operators fail. Although not reflected by standard quality metrics (see next subsection), a wrong hyphenation usually is eye-catching for human readers, which gives rise to a desirable further quality improvement that should be aimed for in future work. The other operator usages are mostly word insertions and deletions (about 25%), synonym replacements (3%), and function words (3%). The context-independent operators of contractions and spelling correspond to only about 1% of all operator applications.

5.3 Quality-Related Analysis

Table 3 also contains information about the text quality before and after generating the acrostic. To algorithmically measure text quality-related effects, we employ a word frequency class analysis and a readability analysis.

The frequency class WFC(w) of a word w relates to its customariness in written language and has successfully been employed for text genre analysis (Stein and Meyer zu Eißen, 2008). Let $\varphi(w)$ denote the frequency of a word in a given corpus; then the Wortschatz[8] defines WFC(w) as $\lfloor \log_2(\varphi(w^*)/\varphi(w)) \rfloor$, where w^* denotes the most frequently used word in the respective corpus. Here we use as reference the Google n-gram corpus (Brants and Franz, 2006) whose most frequent word is "the", which corresponds to the word frequency class 0; the most uncommonly used words within this corpus have a word frequency class of 26. The readability of the text before and after acrostic generation is quantified according to the standard ARI (Smith and Senter, 1967) and SMOG (McLaughlin, 1969) measures, implemented in the Phantom Readability Library.[9] Both measures have been designed to estimate the U.S. grade

[8] wortschatz.uni-leipzig.de, last accessed: June 12, 2014.

[9] http://niels.drni.de/s9y/pages/phantom.html, last accessed: June 12, 2014

level equivalent to the education required for understanding a given text. Hence, larger readability scores indicate more difficult texts. The Automated Readability Index ARI (Smith and Senter, 1967) is designed for being easily automatable and uses only the number of characters (excluding whitespace and punctuation), words, and sentences in the text (delimited by a period, an exclamation mark, a question mark, a colon, a semicolon, or an ellipsis). The Simple Measure of Gobbledygook SMOG (McLaughlin, 1969) includes the number of words with more than three syllables, so-called polysyllables.

$$\mathrm{ARI} = 4.71 \cdot \frac{\mathrm{Characters}}{\mathrm{Words}} + 0.5 \cdot \frac{\mathrm{Words}}{\mathrm{Sentences}} - 21.43 \qquad \mathrm{SMOG} = 1.0430 \cdot \sqrt{30 \cdot \frac{\mathrm{Polysyllables}}{\mathrm{Sentences}}} + 3.1291$$

Of course, the above three measures cannot capture text quality as human judges would perceive it. Still, they have their merits and can indicate interesting trends: On average, the texts after acrostic generation use more common words (cf. the negative Δ WFC) and are easier to read (cf. the negative Δ ARI and Δ SMOG) for almost all acrostic types. Thus, one may argue that the quality is not harmed too much; still some issues like wrong hyphenation are ignored by the metrics (cf. the above discussion of individual operators). A deeper analysis of operator quality and improved quality of paraphrased texts (e.g., further operators or avoiding wrong hyphenations) constitute very promising directions for future work.

6 Conclusion and Outlook

We have presented the first algorithmic approach to acrostic generation. The experiments show that the heuristic search approach is able to generate about 20% of the target acrostics in reasonable time, whereas the producibility of the first letter plays a key role. Our solution successfully combines paraphrasing techniques from Natural Language Processing with a heuristic search strategy from Artificial Intelligence. This way, our problem modeling opens a novel and very promising research direction, and the application of our framework to other paraphrasing problems is the most interesting line of future work. As for the acrostic use case, the resulting text's quality gives the most obvious possibility for improvements. We plan to further analyze better quality measures for the individual operators and to develop more sophisticated operators like changing a text's tense or even anaphora exploitation (Schmolz et al., 2012).

References

Ion Androutsopoulos and Prodromos Malakasiotis. 2010. A Survey of Paraphrasing and Textual Entailment Methods. *Journal of Artificial Intelligence Research*, 38(1):135–187.

Guy Aston and Lou Burnard. 1998. The BNC Handbook. http://www.natcorp.ox.ac.uk.

Colin J. Bannard and Chris Callison-Burch. 2005. Paraphrasing with Bilingual Parallel Corpora. In *Proceedings of ACL 2005*, pages 597–604.

Regina Barzilay and Lillian Lee. 2003. Learning to Paraphrase: An Unsupervised Approach Using Multiple-Sequence Alignment. In *Proceedings of HLT 2003*, pages 16–23.

Regina Barzilay and Kathleen McKeown. 2001. Extracting Paraphrases From a Parallel Corpus. In *Proceedings of ACL 2001*, pages 50–57.

Igor A. Bolshakov and Alexander F. Gelbukh. 2004. Synonymous Paraphrasing Using WordNet and Internet. In *Proceedings of NLDB 2004*, pages 312–323.

Thorsten Brants and Alex Franz. 2006. Web 1T 5-gram Version 1. Linguistic Data Consortium LDC2006T13.

Steven Burrows, Martin Potthast, and Benno Stein. 2013. Paraphrase Acquisition via Crowdsourcing and Machine Learning. *ACM Transactions on Intelligent Systems and Technology*, 4(3):43:1–43:21.

Chris Callison-Burch. 2008. Syntactic Constraints on Paraphrases Extracted from Parallel Corpora. In *Proceedings of EMNLP 2008*, pages 196–205.

Jonathan Chevelu, Thomas Lavergne, Yves Lepage, and Thierry Moudenc. 2009. Introduction of a New Paraphrase Generation Tool Based on Monte-Carlo Sampling. In *Proceedings of ACL 2009*, pages 249–252.

Jonathan Chevelu, Ghislain Putois, and Yves Lepage. 2010. The True Score of Statistical Paraphrase Generation. In *Proceedings of COLING 2010 (Posters)*, pages 144–152.

Paul Clough, Robert Gaizauskas, Scott S. L. Piao, and Yorick Wilks. 2002. METER: MEasuring TExt Reuse. In *Proceedings of ACL 2002*, pages 152–159.

William B. Dolan and Chris Brockett. 2005. Automatically Constructing a Corpus of Sentential Paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing 2005*, pages 1–8.

Christiane Fellbaum. 1998. *WordNet: An Electronic Lexical Database*. MIT Press.

Juri Ganitkevitch, Chris Callison-Burch, Courtney Napoles, and Benjamin Van Durme. 2011. Learning Sentential Paraphrases From Bilingual Parallel Corpora for Text-to-Text Generation. In *Proceedings of EMNLP 2011*, pages 1168–1179.

Juri Ganitkevitch, Benjamin Van Durme, and Chris Callison-Burch. 2013. PPDB: The Paraphrase Database. In *Proceedings of HLT 2013*, pages 758–764.

David Kauchak and Regina Barzilay. 2006. Paraphrasing for Automatic Evaluation. In *Proceedings of HLT 2006*.

Bryan Klimt and Yiming Yang. 2004. The Enron Corpus: A New Dataset for Email Classification Research. In *Proceedings of ECML 2004*, pages 217–226.

Donald E. Knuth. 1986. *The TEXbook*. Addison-Wesley.

David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li. 2004. RCV1: A New Benchmark Collection for Text Categorization Research. *The Journal of Machine Learning Research*, 5:361–397.

G. Harry McLaughlin. 1969. SMOG Grading: A New Readability Formula. *Journal of Reading*, 12(8):639–646.

Donald Metzler and Eduard Hovy. 2011. Mavuno: A Scalable and Effective Hadoop-Based Paraphrase Acquisition System. In *Proceedings of the Third Workshop on Large Scale Data Mining 2011*, pages 3:1–3:8.

Donald Metzler, Eduard H. Hovy, and Chunliang Zhang. 2011. An Empirical Evaluation of Data-Driven Paraphrase Generation Techniques. In *Proceedings of ACL 2011 (Short Papers)*, pages 546–551.

Bonnie J. F. Meyer. 2003. Text Coherence and Readability. *Topics in Language Disorders*, 23(3):204–224.

Bo Pang, Kevin Knight, and Daniel Marcu. 2003. Syntax-Based Alignment of Multiple Translations: Extracting Paraphrases and Generating New Sentences. In *Proceedings of HLT 2003*, pages 102–109.

Marius Pasca and Péter Dienes. 2005. Aligning Needles in a Haystack: Paraphrase Acquisition Across the Web. In *Proceedings of IJCNLP 2005*, pages 119–130.

Judea Pearl. 1984. *Heuristics*. Addison-Wesley.

Chris Quirk, Chris Brockett, and William B. Dolan. 2004. Monolingual Machine Translation for Paraphrase Generation. In *Proceedings of EMNLP 2004*, pages 142–149.

Patrick Riehmann, Henning Gruendl, Martin Potthast, Martin Trenkmann, Benno Stein, and Bernd Froehlich. 2012. WORDGRAPH: Keyword-in-Context Visualization for NETSPEAK's Wildcard Search. *IEEE Transactions on Visualization and Computer Graphics*, 18(9):1411–1423.

Helene Schmolz, David Coquil, and Mario Döller. 2012. In-Depth Analysis of Anaphora Resolution Requirements. In *Proceedings of TIR 2012*, pages 174–179.

Edgar A. Smith and R. J. Senter. 1967. Automated Readability Index. Technical Report AMRL-TR-6620, Wright-Patterson Air Force Base.

Benno Stein and Daniel Curatolo. 2006. Phonetic Spelling and Heuristic Search. In *Proceedings of ECAI 2006*, pages 829–830.

Benno Stein and Sven Meyer zu Eißen. 2008. Retrieval Models for Genre Classification. *Scandinavian Journal of Information Systems (SJIS)*, 20(1):91–117.

Benno Stein, Martin Potthast, and Martin Trenkmann. 2010. Retrieving Customary Web Language to Assist Writers. In *Proceedings of ECIR 2010*, pages 631–635.

Nathan Sturtevant, Ariel Felner, Maxim Likhachev, and Wheeler Ruml. 2012. Heuristic Search Comes of Age. In *Proceedings of AAAI 2012*.

Shiqi Zhao, Cheng Niu, Ming Zhou, Ting Liu, and Sheng Li. 2008. Combining Multiple Resources to Improve SMT-Based Paraphrasing Model. In *Proceedings of ACL 2008*, pages 1021–1029.

Shiqi Zhao, Xiang Lan, Ting Liu, and Sheng Li. 2009. Application-Driven Statistical Paraphrase Generation. In *Proceedings of ACL 2009*, pages 834–842.