Technische Universität Berlin
Fakultät IV - Fakultät Elektrotechnik und Informatik
Fachgebiet Datenbanksysteme und Informationsmanagement

Project Report
Generating Acrostics via Paraphrasing and Heuristic Search
DBPRO - Database Projects (WS 2014/2015)

Supervisor:
Johannes Kirschnick

Authors:
Bruno Soares Fillmann ()
Fernando Bombardelli da Silva (bombardelli.f@mailbox.tu-berlin.de)
Jürgen Bauer ()
William Bombardelli da Silva (wbombardellis@mailbox.tu-berlin.de)

February 2, 2015

Contents

Abstract

ascasdas

1 Introduction and Motivation

lnmkm

2 Generating Acrostics via Paraphrasing and Heursitic Search

2.1 Problem Definition

2.2 Modeling as Search Problem

2.3 Cost Measure

2.4 Operators

2.4.1 Word Insertion or Deletion

The idea around this operator is to insert words in the text or delete words from it, in order to insert new letters and accomplish the goal acrostic or to remove words and change the position of words inside the text.

To ilustrate the execution, consider the following text[1]:

Ah ja, ich heisse Frederik Hoske und ich bin 13 Jahre. *Ich kann nicht vorstellen*, weil ich kaum Deutsch sprechen kann. Trotzdem versuche ich es. Ich habe zwei Geschwister Mein Bruder der 16 Jahre alt ist und meine Schweseter ist elf.

After inserting the word *"mir"* in the sentence *"Ich kann nicht vorstellen"* in the first line and after breaking a line right before *"Trotzdem"* the algorithm can reach the acrostic *amt*. Note that the insertion of *"mir"* was crutial for the result, once that the letter *m* was not there.

Ah ja, ich heisse Frederik Hoske und ich bin 13 Jahre. *Ich kann mir nicht vorstellen*, weil ich kaum Deutsch sprechen kann. Trotzdem versuche ich es. Ich habe zwei Geschwister Mein Bruder der 16 Jahre alt ist und meine Schweseter ist elf.

The Word Insertion or Deletion operator takes as input a text. Then first it tries to insert a new word in each space and second tries to remove each word of the text. The condition to insert a new word w in the *i-th* space of the text is that w has to fit the context around the *i-th* space. It means that from the set of all possible words of the language, only a restricted subset can be inserted in this place. More specifically, the algorithm starts by taking for each space in the text n words around it as context − In our implementation in this context $n = 4$. This is a so called n-gram, an array of words. After this, the n-gram just taken is sent to the context database (which is in this implementation the NetSpeak API [2]), that returns the possible

[1] This text was adapted for didactic purposes from http://cornelia.siteware.ch/blog/wordpress/2008/11/03/sich-vorstellen-horverstehen. Access in January, 2015

words that could be inserted in the required space. For each of these possible words a new version of the text is created with the word inside.

Analogously, for each word w in the text a n-gram including the words around it is created − In our implementation we take two words from each side, so here $n = 5$. w is then taken out of the n-gram, which is tested against the context database to check whether this n-gram is frequent enough in the language. If the answer is positive a new version of the text without w is created. Our implementation allows the adjustment of the minimum frequency cited above, but we set it to zero, so a broader set of deletions is executed.

The queries to the context database are made in form of HTTP requests to the netspeak web service using the NetSpeak API.

2.4.2 Synonyms

The synonym operator has the goal of changing words in the text for other words, which have similar meaning. In general the operator takes a text as input and generates a set of new texts, in which each text has a word replaced by a synonym that may eventually contain more than just one word.

In order to perform the replacements it is required a synonym dictionary, which is know as thesaurus. In our implementation we used Open Thesaurus [3], which is available for download for free. This data source is available as a plain text file, but as the dictionary is accessed many times during the execution of the algorithm, it easily becomes intractable to handle a text file as a database.

To solve this problem we decided to use a NoSQL database server [4], namely, Redis. Redis is an open source advanced key-value pair cache and store [5]. Into the database server we load once the data from the thesaurus in a structured way where, every word is added as a key that points to a set of synonyms. Thus can the application easily and efficiently find similar terms for a given word only by accessing this key.

Naturally it is then required that the Redis server is running and listening to requests when the application runs, and that it has been once loaded by our script with the data from the dictionary.

[EXAMPLE]

PROS

CONS

2.4.3 Line break

The fastest and most basic operation is the line break. A line break can be applied in two cases:

- After a word when the line length lies in the $[l_{min}, l_{max}]$-window, given by the line length constraints.
- After the end of a sentence.

After performing a line break, the lines following the line break have to be aligned again to satisfy the line length constraints.
For this task we apply a greedy word wrap algorithm, which works as follows: we split the text into words, put the words on the line as long as there is free space, if there is no free space left, we continue with the next line.
When applying the greedy word wrap algorithm we have to ensure that there is no word of length > 20 in the initial text. Otherwise it might happen, that the minimal line length constraint is not fulfilled.
Identifying the end of a sentence in general is a difficult problem. One reason for this is that a period might occur in several contexts, e.g.

- abbreviations (Prof., Dr., d.h., z.B., ...)
- ordinal numbers (der 26. April, Joseph II., 2. Auflage, ...)
- numbers (10.1312, 192.11301, ...)

Another issue is that there is a wide variety of punctuations which could mark the end of a sentence. These punctuations include question marks, exclamation marks, ellipses, semi-cola, cola.
To overcome these problems we make use of a sentence-splitter library, called Sentrick (cf. [6]).

2.4.4 Hyphenation

Related to the line break are hyphenations. A hyphenation is applicable if the line after hyphenating (and line breaking) has a length of at least $l_{min} = 50$.
For hyphenating a word we employ a re-implementation of Knuth's hyphenation algorithm in TEX (cf. [7]).
After the hyphenation, the text following the hyphen has to be aligned again to satisfy the line length constraints.
Analog to the line break operation, in order to rearrange the lines we apply a greedy word wrap algorithm.

3 Evaluation of the Results

4 Summary of Findings

to better: size of ngram, other context databas, degenerating into breadth first, webservices slow, databases not good

References

[1] Benno Stein, Matthias Hagen, and Christof Bräutigam. *Generating Acrostics via Paraphrasing and Heuristic Search.*
In Junichi Tsujii and Jan Hajic, editors, 25th International Conference on Computational Linguistics (COLING 14), pages 2018-2029, August 2014. Association for Computational Linguistics.

[2] Martin Potthast, Martin Trenkmann, and Benno Stein. *Netspeak: Assisting Writers in Choosing Words.* In Cathal Gurrin et al, editors, Advances in Information Retrieval. 32nd European Conference on Information Retrieval (ECIR 10) volume 5993 of Lecture Notes in Computer Science, pages 672, Berlin Heidelberg New York, March 2010. Springer.

[3] *Synonyme - OpenThesaurus - Deutscher Thesaurus.* Available on: <`https://www.openthesaurus.de`>. Accessed in: Januar 2015.

[4] Jing Han; Haihong, E.; Guan Le; Jian Du. *Survey on NoSQL database.* Pervasive Computing and Applications (ICPCA), 2011 6th International Conference on, pp.363,366, 26-28 October 2011.

[5] *Redis.* Available on: <`http://redis.io`>. Accessed in: Januar 2015.

[6] `http://sourceforge.net/projects/sentrick/`

[7] `http://sourceforge.net/projects/texhyphj/`

A Appendix