

Statistics and Graphotactical Rules in Finding OCR-errors

14 january 2000

Stina Nylander
stina@stp.ling.uu.se

Language Engineering Programme
Department of Linguistics
Uppsala University

Supervisor: Anna Sagvall Hein
Assistant supervisor: Jussi Karlgren

Abstract

This thesis describes two experiments in finding errors in optically scanned Swedish without relying on a lexicon. First, statistics were used to find unexpectedly frequent trigrams and correction rules for these cases were created. The rules were then tested and compared to a hand corrected version of the test text. Secondly, Bengt Sigurd's model of Swedish phonotax was adapted to spelling and used to detect words with graphotactically illegal beginning or end.

The rules generated in the statistical experiment proved to be very specific. They performed well on the training material but did very few corrections on new text material. The graphotax did not perform as well as the statistical rules did on the training material, but outscored them by far on new text. In a small text material the graphotactical rules managed to find about 30% of the errors. Graphotax was thus considered as the better approach in finding OCR errors.

A graphical correction tool was created for detection of graphotactical errors. The tool displays every occurrence of an error string at the same time and gives the user the possibility to give different corrections to each occurrence.

This work shows that it is possible to find errors in optically scanned text without relying on a lexicon, and that graphotactical word structure can provide useful information to the correction process.

Acknowledgements

This work has been funded by the Swedish Institute of Computer Science (SICS). I would like to thank my supervisor from Uppsala university, Anna Sagvall Hein, for help and support during the work. Many thanks to my assistant supervisor at SICS, Jussi Karlgren, who encouraged me during the work, always had a solution when problems came up, and always had a positive way of seeing things. Fredrik Olsson has given me invaluable help with the Perl programming, and Anders Gabrielsson and Nils af Geijerstam have proofread this report.

Contents

1	Introduction	1
1.1	The Task	1
1.2	Outline of this Thesis	2
2	Background	3
2.1	OCR	3
2.2	OCR Errors	4
2.3	Proofreading	4
2.4	The Need for OCR — Now and in the Future	5
3	Earlier Work	7
4	Statistical Methods	9
4.1	The Errors	11
4.2	The Rules	12
4.3	Results	12
5	Phonotax and Graphotax	15
5.1	The Rules	19
5.2	The Errors Found	20
5.3	Results	21
6	Implementation	23
6.1	Technical Description	23
6.2	Graphical Interface	24
6.3	Subroutines	24
6.3.1	errorfinder	24
6.3.2	makeConcordance	24
6.3.3	fromhtml	25
6.3.4	makeChanges	25
6.3.5	tohtml	25
6.3.6	errorDialog	25
6.3.7	changeDialog	25
6.3.8	naesta	25

6.3.9	slut	25
6.3.10	sluta	26
6.4	Error Handling	26
6.4.1	External Error Handling	26
6.4.2	Internal Error Handling	26
6.5	Possible Improvements	27
7	Discussion	29
8	References	31

Chapter 1

Introduction

Optical character recognition (OCR) is a technique for moving text resources from paper medium to electronic form, something that is often needed in our computerised society; companies, research institutions and authorities want to make old material machine readable or searchable. Unfortunately, it does not get us all the way. With good paper originals, OCR can achieve 99% of the characters correctly recognised but the result will still contain in average one error word per 20 words which means 5% incorrect words or about one error per sentence (Kukich, 1992b). Depending on the application of the optically scanned text, large post processing efforts may be required. Since OCR is often used to move large amounts of text to electronic form, the proofreading is a task both demanding and dull. This makes the need for good tools of spell checking and correction strong and urgent.

Most spell checkers and OCR post processing systems are lexicon based. A word list of reasonable size is used to match against the text, and any word token not in the list is presented as a possible error. Probability scores or similarity measures are then used to generate correction suggestions.

Here, I will mostly concentrate on the error finding process. I want to find ways of proofreading text without relying on a lexicon. Instead I will try to define rules that identify character sequences that are unlikely to be correct word tokens. I made two experiments: using statistical methods to find unexpectedly frequent character sequences, and using phono- or graphotactical rules to find unlikely character combinations. The resulting rules have been integrated in a proofreading tool.

1.1 The Task

I was given the assignment to develop a tool for correction of optically scanned Swedish text. The tool should not use a lexicon but statistics or phonotax to detect errors. I was to find out which one of the two methods would be the best to use, to decide if the tool should be automatic or interactive, and also

to implement my results. My main goal is to develop methods for proofreading optically scanned Swedish, and to design the software in a way that it will be easy to plug in data or rules for a new language or a new domain. Obviously these results can be generalised for all kinds of proofreading tasks: e.g. handwriting recognition or dictation tasks. The experiment has been conducted on English and Swedish material.

The work was carried out within the research project Digital Libraries, initially funded by Svenska IT-institutet, at the Swedish Institute of Computer Science (SICS).

1.2 Outline of this Thesis

In this report I discuss how to find errors in optically scanned text. Chapter 2 gives some background information on OCR (subsection 2.1), the errors specific for OCR and their causes (subsection 2.2), proofreading optically scanned text (subsection 2.3), and the need for OCR (subsection 2.4).

In chapter 3, I will make a short survey of recent research in the field to show which are the central problems and how they are resolved in different applications.

In chapter 4, I will describe the statistical experiment I did; the methods used, and the decisions made. Subsection 4.1 treats the errors in the material, and subsection 4.2 the rules created. Subsection 4.3 then presents the result.

Chapter 5 deals with the phonotactical experiment. I will start with a short explanation of phonotax, then I will describe the phonotactical model that I used for the experiment, and the modifications I made in the model during the experiment. Subsection 5.1 describes the phono- or graphotactical rules and subsection 5.2 the errors the rules managed to find. Subsection 5.3 then presents the result.

Chapter 6 describes the implementation that resulted of the experiments. In subsection 6.1 I will describe the functions of the program, in subsection 6.2 the graphical interface and in subsection 6.3 I will give a description of each subroutine. Subsection 6.5 finally deals with how the program could be improved.

In chapter 7 I discuss my results.

Chapter 2

Background

2.1 OCR

Optical character recognition is, in fact, the conversion of a black and white image to an electronic text document. The scanner digitalises the image into a list of black and white dots. The OCR software then tries to map groups of black dots to stored shapes. It looks for horizontal and vertical lines, semi-circles in isolation, and it tries to discard stray dots, fill in gaps, shrink or enlarge shapes etc. These techniques are called *feature extraction* (Bos et al. 1993). The shortcomings of these techniques can explain why some errors occur during the recognition process. For example is a *d* a semi-circle and a vertical line, and so is *cl* and *l*. The recognition easily slips if the letter is not properly filled in, if the contrast of the paper original is bad or if the font size is very small or very large. The errors that result from such slips in the recognition can be very difficult to correct without access to the paper original. For example *liciriograf* is not a Swedish word, and to guess that it is a misrecognition of *homograf* is not very easy without a quite large context, but *li* instead of *h*, *c* instead of *o*, and *iri* instead of *m* are all quite normal recognition errors.

The feature extraction is completed with letter and letter-pair frequencies, to make it possible for the system to differentiate between doubtful letters like the '0' in '10053' and the O in LOOSE. Lexicon techniques are used too, in doubtful cases a sequence found in lexicon is preferred to one that is not (Bos et al. 1993).

Commercial OCR software — like OmniPage from Caere or Textbridge from Xerox — supports several languages. The user tells the program which language the document to be scanned in is containing. Knowing which language it is working with, the software also knows which character set to use. If the language is set to English the software will not map any images to the Swedish vowels *å*, *ä* or *ö*. However, documents containing several languages often causes trouble, even if some products pretend to handle multilingual documents. Some OCR products also have an interactive training function where the user can map an

image to a character if the program fails. The image and the character are then stored and the program can use the information at a later point in the document, or in another document (Dahlqvist, 1997).

2.2 OCR Errors

Scanning errors are usually divided into two groups, *non-word errors* (mostly called only errors) and *real word errors*. In the *non-word error* group we find strings that are non-words in the language in question and in the *real word error* group we find strings that are correct words in the language in question but not the word found in the original text. For example, if the text *John found the man* is rendered as *John fornd he man* by the OCR device, then *fornd* is a non-word error and *he* is a real word error.¹

Very specialised OCR systems, like bank check scanning, can have an error rate as low as one character per million. More general systems normally have an error rate of 1-2% of the characters for printed text with arbitrary font (Dahlqvist, 1997).

The main cause of recognition errors is graphical similarity. The image is not clear enough to make it possible for the OCR device to draw the right conclusion. Then two things can happen: the OCR device cannot classify the character at all and outputs a default character (usually ~), or the character is wrongly recognised (Dahlqvist, 1997). If the character is wrongly recognised, the OCR device chooses a similar shaped character that is more highly frequent in the language or that gives a string in lexicon. Examples: **arguinent** argument, **teature** (feature), **rnean** (mean), **sernantics** (semantics), **systemet** (systemet), **textförståelse** (textförståelse), **disambiqueras** (disambigueras). Since the graphical errors by definition are difficult to detect by ocular scanning through the text, the proofreading by hand gets even more difficult. The visual difference between *bodü* and *body* is very small and easy to miss when reading fast. Other problems are print quality, font, font size, and the age of the original that sometimes produce errors that make it impossible to guess the original word like **appTijdo-tiL-s** (approaches.) or **Umt** (that).

Another group of errors that occurs in optically scanned text is split errors; spaces are inserted in a word and produce a number of strings, many of them incorrect: **pronunc i at ion** (*pronunciation*), **i nt e** (*inte*), **öre I i gger** (*föreligger*). However, many or even most of the graphical and split errors still produce string tokens that are unlikely or impossible words in the language under consideration.

2.3 Proofreading

Proofreading optically scanned text differs from proofreading typed text. Most people make errors while typing a longer text — fingers slip on keys or you

¹Example from Tong & Evans, 1996.

misspell a word when thinking too fast — but the number of errors are normally much lower than the error level of optically scanned text. Studies made in typewriting show that expert typists have an error rate of 1% on word level and novice typists 3% when typing a given text from a written original (Grudin, 1983). In a different situation the error rate is different. The Telecommunications Device for the Deaf in the US provides a service where a hearing person can communicate with a deaf person through a human operator that types the spoken message and sends it to the deaf person. The deaf person can then respond with a text message that is read to the hearing person by the operator. A study on these typed messages show an error rate of 5-6% on word level on text produced from speech input and in a real time communication situation (Kukich, 1992a). With 99% correct character recognition and one word error per 20 word (Kukich, 1992b), OCR thus does not perform better than a good typist. And in many cases OCR achieves less than 99% correctly recognised characters.

Facing a proofreading task knowing that the text to be corrected not only is very long — optical scanning is mostly used on large materials — but also contains a lot of errors is a challenge. One possible solution to this is that text with a lot of errors should not be proof read linearly — that takes too much time and energy. A good tool for correction should be able to show all occurrences of a possible error at the same time, with a small context, and let the user decide which are wrong and easily correct them. Showing all occurrences at the same time gives the proofreader the possibility to judge if all occurrences really are wrong, one or two might be the right use of an unusual spelling, or be an unusual acronym. It also saves time since the same error will not have to be examined twice. The context helps the proofreader to make the right correction, sometimes it is very difficult to see which correct word lies behind the misrecognised string. It is also important to be able to give different corrections to different occurrences of an error string.

Using an ordinary spell checker or another lexicon based proofreading package becomes a very slow process while working with optically scanned text. Lexicon based systems define errors through a list with words that are correct and words not in the list are presented as possible errors to the proofreader. These systems produce a lot of possible errors that in fact are correct words, and this — combined with the large number of errors in the optically scanned text — makes the proofreading a very slow and often irritating process. The false alarms could be reduced by using another error definition. A lexicon defines what is right, while phonotax, for example, defines what is wrong.

2.4 The Need for OCR — Now and in the Future

At the present time, optical scanning is the easiest way to convert text from paper to electronic form — it is fast and gives a reasonable good result if the

paper originals are good. The gain in time depends of course on the quality of the OCR, a lot of errors demand a lot of post processing which is very time consuming. If more than 5% of the words are incorrectly recognised (which corresponds to about 1% of the characters incorrectly recognised), the gain in time is getting very small compared to a good typist (Bos et al. 1993).

Optical scanning is still very useful in linguistic research, books, newspaper material and other texts can be made searchable without hours of typing. In many cases the fact that texts become searchable is so important that this compensates for the errors. The nineties has been the decade in which most authorities and companies have computerised their administration. Now there is a need to convert old registers and archives to machine readable formats, or create searchable indexes over the old material. This is done using optical scanning as method: the archive of the Swedish Parliament is using optical scanning to create machine readable and searchable indexes over minutes and other documents, hospitals are converting medical journals into machine readable forms, companies are scanning invoices and incoming mail. In a short term — five or ten years from now — I can see a big need for getting text moved from paper to electronic form, and optical scanning is at the moment the best way to do it. It is a way to make large materials searchable for research purposes or to make it accessible for the public.

In a longer term — 20-30 years from now — maybe the need of optical scanning will decrease since most of the text material that will be produced from now on will be produced in electronic form and electronic mail will be used more than paper mail. On the other hand, during this work I have noticed that storage formats tend to change faster than people lose interest in the stored material. In a period of 10 to 15 years the standards of both hardware, software and storage formats change so much that it can be very difficult to read old material. It can also be almost impossible to find. The easiest way to get old material in machine readable format might still be to scan the paper copy, even if there is an electronic version. And there will always be people, researchers or others, interested in texts that no one has worked with before. Hopefully the performance of the OCR systems will improve so that the need of proofreading will be smaller.

Chapter 3

Earlier Work

Most approaches to the correction of scanning errors are lexicon based and uses non-word errors as error definition. Errors are detected by searching the text for words that do not appear in lexicon, and any word not in lexicon is presented as a possible error. This leads to many false alarms, since a lexicon never can cover everything. Many correct words and proper names not in the lexicon will be presented as errors by the system. Additionally, this method will not find the real word errors.

Different methods are used to generate the correction suggestions, for example noisy channel models (Kernighan et al., 1990), confusion sets (Golding & Schabes, 1996), and edit distances combined with word frequencies (Tillenius, 1996).

The only application that claims to be not fully lexicon based I have found in the research for this thesis is the experiment with Bloom filters made by Domeij et al. (1994). They did not use a lexicon in the ordinary meaning of the word. A Bloom filter is a vector of Boolean values and a number of hash functions, which means that it is basically a way of storing and searching a lexicon. To find out if a word is a part of the lexicon, the hash values of the word are calculated and compared to the vector. This means that if a correct word has the same hash values as an incorrect word, they will both be regarded as correct words by the system. The difference between this and a lexicon based system is thus very small. It finds no more real-word errors than does a regular lexicon-based model, and additionally misses a small number of false positives.

To find real word errors — i.e. scanning errors that result in another correct word — sequences of parts of speech are evaluated for likelihood of occurrence, and unlikely sequences are marked as possible errors (Meknavin et al., 1998; Tong & Evans 1996; Huismann, 1999). As can be expected, this sort of processing calls for language specific knowledge resources and in some cases subject domain knowledge too.

Some systems are not really OCR post processing systems but designed for very specific purposes, for example correcting input to a specific machine translation program (Hogan, 1999). Others are very tightly connected to the

OCR process and are designed to function with OCR devices that give several recognition alternatives to a sequence of characters. The post processing system then evaluates the likelihood of the different alternatives (Lee et al., 1996).

The research performed shows that OCR post processing problems are highly language specific: Meknavin et al. show that one of the biggest problems working with Thai is to find the word boundaries (1998), while those working with English put the largest effort in detecting the errors (Tong & Evans, 1996; Golding & Shabes, 1996), or providing good correction suggestions (Takahashi et al., 1990). Lee et al. (1996) argue that the Korean writing system is syllable based and that recognition and error correction therefore should be syllable based rather than character based. Hogan points out that when you work with minority languages, in his case Haitian Creole, it is not likely that you even use OCR software developed for your language, something that makes post processing even more necessary (1999).

Chapter 4

Statistical Methods

The starting hypothesis of this study was that a difference in frequency for a sequence of characters, an n-gram, between correct text and optically scanned text would indicate that the same n-gram did not always come out correctly from the OCR process. The work described below concentrates on sequences of three characters, trigrams, that are more frequent in optically scanned text than in correct text.

The NoDaLiDa conference proceedings were selected as experimental material. The proceedings contain both correct text as provided by the author in machine readable form and non-corrected optically scanned text. To get material for the frequency computation two reference texts were put together, one with correct text and one with optically scanned text containing errors, each containing about 75 000 words. Since I only have access to large quantities of optically scanned Swedish and very little Swedish text provided in machine readable form, it was not possible to get enough frequency data for Swedish trigrams. The statistical experiment therefore has been done on English text.

A simple change in frequency could not be used as the only criterion to find suspect trigrams since the two texts used are not parallel texts — i.e. the optically scanned text is not the correct text that has been scanned in — but comparable texts — i.e. the same genre (scientific articles) in the same subject domain (computational linguistics). To be considered as suspect, a trigram also had to show a difference in position in the frequency list, i.e. a changed frequency in relation to the other trigrams in the frequency list. A difference in position and frequency between the two texts is not necessarily due to recognition errors but might be an indication that a trigram has become more or less frequent in relation to other trigrams. A frequency list of every occurring trigram was generated for both correct and optically scanned text. A trigram was considered as suspect if it only occurred in optically scanned text, or if it had climbed at least a certain threshold value (set to 100 in this experiment) in position, i.e. had become more frequent in optically scanned text in relation to other trigrams. I made the assumption that 75 000 words is enough to establish a trigram frequency list that is stable enough to serve as

standard. Consequently the trigrams in the optically scanned text would have approximately the same order in the frequency list, even if the frequencies would differ. Both frequency lists contained about 20 000 trigrams but only about 10 000 occurred in both lists so the assumption did not hold. Generating the rules was thus time consuming since the list of suspect trigrams was very long and contained many correct trigrams.

Two optically scanned papers were hand corrected to obtain testing material: *How Close Can We Get to the Ideal of Simple Transfer in Multi-lingual Machine Translation (MT)?*, henceforth NODA89-09 (Andersen, 1989), and the first part of *A self-extending lexicon: description of a word learning program*, henceforth NODA85-06 (Ejerhed & Bromley, 1985).

Since it is necessary to be able to count the number of errors automatically in studies like this — proofreading and counting errors by hand is simply too costly in time — a simple error measure was adopted: the number of errors in an optically scanned text is the difference in word count between the optically scanned text and the correct text, plus the number of strings that only appeared in the optically scanned text. The real word errors would not be included in the resulting number of errors and a split error would be counted as the number of parts the original word was split into. In this study, errors have been defined as differences between the original and the optically recognised text. Spelling errors in the original that has been recognised as the same spelling error have thus not been counted among the errors, but in a real proofreading situation this is a group of errors that must be considered and corrected.

Two sets of rules were generated for each article: one set of rules that changed a trigram to a string of arbitrary length and one set that changed a string of arbitrary length into another string of arbitrary length. Each set of rules were then used to correct both the article that had been used to generate the rules as well as the other, to see if the rules were useful in a context different from the one they had been generated in.

To support the generation of the rules that rewrite trigrams, a graphical tool was created in Perl/Tk that compared the n-gram frequencies of the correct text and the optically scanned text, generated a list of n-grams that showed frequency differences, for each n-gram showed a concordance of all the occurrences of the n-gram. The rules that rewrite longer strings were generated by hand with a concordance over the optically scanned text as support.

The number of errors was counted by means of a Perl program before and after correction to estimate the performance of the rules. The number of errors generated in the correction process was counted separately to keep track of over-correction. Strings that appeared only in the version corrected with the rules, neither in the correct version nor in the optically scanned version were considered as generated errors. Over-corrections that result in strings — words or non-words — already present in the correct text or the optically scanned non-corrected text will thus not be discovered.

Error group	error example	correct word
Missing word		
Truncation	interlin	interlingua
Inserted upper case letter	langUage	language
Inserted digit	word5	words
Inserted special character	ta%ks	tasks
Foreign character	Blaberg	Blåberg
One misrecognised letter	agteement	agreement
Split word	syn t a c t i c	syntactic
Missing space	propositiondu	proposition du
Misrecognised foreign word	hande	hende
More than one letter misrecognised	ligwe	figure
Error due to rules or code in the text	da-isframe	da_isframe
Impossible	u,jciids)q	words),
Miscellaneous	are.,	are:

Table 4.1: Error groups with examples

4.1 The Errors

A categorisation of the errors in NODA89-09 and NODA85-06 showed a great variety among the errors and big differences between the two texts. See table 4.1 for an overview of the error groups with examples. Truncation, foreign characters, inserted upper case letters or digits, inserted special characters, split words, missing words, and one misrecognised letter are groups of errors that could be found in both texts, but the number of errors in each group differed a lot. In NODA89-09 errors due to rules or code (39) and missing words (26) were the most common errors, while in NODA85-06 it was the impossible errors (33) and one misrecognised letter (27). NODA85-06 had a lot of split word errors (17), few foreign character errors (6) and almost no truncation errors (1), while NODA89-09 had a lot of foreign character errors (15) and truncation errors (14) but few split word errors (5).

A part of these differences might be explained by differences of the text originals: NODA89-09 has a better print quality and proportional font, while NODA85-06 has bad contrast and courier font. A clean paper original with good contrast makes it of course easier for the OCR device to recognise the characters. Courier is a monospaced font, which sometimes puts so much space between the characters that the OCR device inserts a space character. A type of errors that is very common in NODA85-06 but does not occur at all in NODA89-09 is misrecognised commas. As much as 22 commas are misrecognised for *g*, *q*, *y*, *p*, *s* or the point.

4.2 The Rules

The rules generated had the following form: `trigram -> string` and `string -> string` where `trigram` is a three character sequence (may include space) and `string` is a sequence of arbitrary length (may include space). Example of trigram rules:

```
5rd->ord
5ti->sti
gUa->gua
r%a->rsa
1hi->Thi
```

and rules that rewrite longer strings (some of the strings include spaces):

```
byWer->bygger
ionaxy->ionary
  pye-> pre
  gAr -> gâr
optd->opt'e
```

Generally, there were fewer trigram rules generated than rules treating longer strings, mostly due to the fact than an error often affects more than three characters. And if that is not the case, the correction even more often concerns more than three characters. A small context is often needed to avoid that the rule is applied in cases where it should not be. For example the trigram *ern* is wrong when *semantics* has been recognised as **sernantics** but perfectly correct in *internal*.

Another problem with correction rules is that the same correction is not always correct to the same incorrect n-gram. For example *axy* in *dicitonaxy* should be changed to *ary* to get *dictionary* while in *syntaxy* it should (in this case) be change to *ax*, to get *syntax*.

4.3 Results

The corrected version of NODA89-09 contained 2484 words, while the optically scanned version contained 2469 words. The number of errors counted before correction was 164. By means of the rule generating tool, 47 trigram rules were created (in some cases several rules to correct the same error, for example both the rules `w5r->wor` and `5rd->ord` were created due to the error string `w5rd`). The rules made 44 corrections and 11 over-corrections which means that after correction, the text contained 20% less errors. Applied to NODA85-06 the rules made two corrections and two over-generations, and thus left the text with the same number of errors as before correction. With the help of a concordance program 52 rules that changed a string of optional length into another were generated by hand. These rules corrected 69 errors (43%) and created 26, and thus left the text with 26% less errors after correction. Applied to NODA85-06

Article	OCR word count	Correct word count	errors	rules	corr	new errors	improvement
NODA89-09 NODA85-06	2469	2484	164	$3 \rightarrow n$		11	20%
				47	44		
NODA89-09 NODA85-06	2469	2484	164	$n \rightarrow n$		26	26%
				52	69		
NODA85-06 NODA89-09	1822	1750	322	$3 \rightarrow n$		5	6%
				36	25		
NODA85-06 NODA89-09	1822	1750	322	$n \rightarrow n$		10	13%
				76	52		

Table 4.2: Rule sets and results.

the rules corrected 26 errors and created 4; a total of 14% of the errors corrected. See table 4.2 for an overview.

The corrected version of NODA85-06 contained 1750 words, while the optically scanned version contained 1822. The number of errors before correction was 322. By means of the rule generating tool, 36 trigram rules were created. The rules made 25 corrections and 5 over-corrections which means that the text contained 6% less errors after correction. Applied to NODA89-09 these rules neither corrected any errors nor created any, and thus did not do anything useful. 76 rules that changed a string of optional length into another were created, but the rules only made 52 corrections and created 10 new errors. Applied to NODA89-09 the rules made no corrections and created no new errors.

The tests described above show that while rules based on observed frequencies of character sequences do provide a noticeable improvement on the training material and presumably will be useful for proofreading a given text, they are too specific for use on other material. When it comes to the trigram rules the problem could be that the trigram as context is too small and that many errors affect more than three characters, but even if the rules that treated longer strings worked a little better on unknown errors, they too were much too text specific. All the rules generated, deal more or less with a given error in a given word, which is even more true for the rules that rewrote strings of optional length: the longer the string that is rewritten, the less generic the rule. This approach needs a very large text material to generate the rules from and a huge number of rules to be of any significant use in correcting new texts.

Chapter 5

Phonotax and Graphotax

Phonotax is another term for phoneme distribution and is a part of the phonological description of a language (Sigurd 1965). The phonotax obviously differ between languages, e.g. Swedish has a lot of long final consonant clusters while Italian has none (Garlén, 1988). There are different levels of phonotax: syllable phonotax, morpheme phonotax and word phonotax. A common distinction to make in phonotax is between primary and secondary sequences. Primary sequences are found in root morphemes and basic word forms and secondary sequences in compounds, inflected and derived words (Garlén, 1988). In an exhaustive phonotactical description of Swedish it would be necessary to take vowel length and word stress into account (Garlén, 1988). This experiment is text oriented and will discard phoneme length and stress. Instead of phonemes I will use the grapheme sequences representing each phoneme, thus a graphotactical model of Swedish. Instead of trying to find error strings by computing frequency data and compare correct text with optically scanned text, I used Bengt Sigurd's model of Swedish word phonotax (1965) to detect phonotactically illegal strings.

Sigurd has made tables over legal initial and final phoneme sequences for Swedish words. He makes the reservation "primary" about the final sequences and treats only basic forms. He excludes compounds, derived words and inflected words. Swedish words can by derivation and inflection get very long final consonant clusters, where *västkustskt* is a standard example, the noun *västkust* with an adjectivising suffix *-sk* and the agreement marker *-t*.

Sigurd lists, in his tables, both the possible combinations of consonants and which vowels that can follow each consonant combination respectively. For example *spl* can only be followed by the vowels *e* and *i*, *tv* only by *i, e, ä, a, å, u* while for example *tr* and *sl* can be followed by any of the nine Swedish vowels. Sigurd lists 55 initial consonant combinations (including \emptyset). When the model was adapted to graphemes, i.e. the phonemes /f/, /j/, /s/ and /ç/ were replaced by all the different Swedish spellings (see table 5.1), the number of legal initial consonant combinations increased to 63. The SUC corpus (Källgren, 1990) was used to check which vowels that followed the different spellings of the phonemes

Phoneme	Spellings
/ʃ/	ch, sch, sh, sj, sk, skj, stj
/ç/	k, kj, tj
/j/	dj, g, gj, hj, j, lj
/s/	c, s, sc, z

Table 5.1: Swedish phonemes with many different spellings.

in the table, *stj* being followed only by *ä*, *ch* by *a, o, e, i*, etc. Sigurd does not treat words that begin with a sequence of vowels, so the possible initial vowel sequences were extracted from the SUC corpus. This added 22 to the total number of initial character sequences (see table 5.2). Some of Sigurds initial sequences that had their only occurrence in a very rare or archaic word that is not in the latest edition of *Svenska Akademiens Ordlista* (1999), like *vro-* in *vrok*, or that has one single example word, like *spry-* in *sprygel* and *vrö-* in *vrövel* has been eliminated from the list of legal initial sequences. In total that gives around 530 initial consonant sequences + vowel or initial vowel sequences.

Sigurd has 102 primary final consonant combinations. To this has been added 11 combinations to cover the Swedish final doubling of consonants after short vowel (*b, d, f, g, l, m, n, p, r, s, t*) and *-x* and *-xt* as final consonant clusters since *x* is realized as two phonemes, /k/ and /s/, and is thus as letter not a part of Sigurds description (In fact, Benny Brodda even says that the letter *x* has no place in a phonological description of Swedish (Brodda, 1979)). The phoneme sequences containing the phonemes /ʃ/ and *ng* in final position have been adapted to the Swedish spelling but not given more sequences than in Sigurd's description. The legal final vowel clusters were listed empirically by extracting all words from SUC that ended in a sequence of vowels. See table 5.3. With these adaptations made, the model could be used to find strings beginning or ending with illegal consonant clusters.

A Perl program that output possible error strings was created. Strings with graphotactically illegal beginning or end, strings containing only consonants, both digits and letters, both upper and lower case letters and strings containing punctuation marks at other positions than final were considered possible errors. A correction tool was created with Perl/Tk that, given the list of possible errors, showed all occurrences of a possible error word, let the user suggest a correction, mark which occurrences that should be corrected and correct them.

A Swedish corpus of optically scanned text from the NoDaLiDa proceedings, containing 71 000 words was put together. A list of abbreviations was made to be used as filter in the error finding process. (Abbreviations from the list of possible errors, from *Svenska Akademiens ordlista* and from *Nusvensk Frekvensordbok*)

Two shorter articles, *Inte bara idiom* henceforth NODA83-03 (Allén, 1983) and *Händelsestyrd textgenerering* henceforth NODA89-03 (Gustafsson, 1989), 2200 and 1500 words respectively, were corrected by hand to serve as standard

Initial vowel	Following vowel	Example
a	a	Aalto
	o	aorta
	u	augusti
	e	aerodynamik
	i	aids
	y	ayatollah
o	a	oas
	o	oordning
	u	outtalad
	å	oåtkomlig
	e	oerhörd
	i	ointaglig
	ä	oändlig
	ö	oönskad
å	a	åar
e	u	Europa
i	a	iakttaga
	o	iordningställa
	ö	iögonfallande
y	o	yoghurt
	u	yuppie
	e	yen
ö	a	öar

Table 5.2: Word initial vowel sequences

vowel	Final vowel	Example
a	o	kakao
	y	display
o	a	roa
	u	Guillou
	e	oboe
	y	playboy
u	a	sjua
å	a	tvåå
e	a	Korea
	o	video
	u	EU
	y	hockey
	å	Luleå
i	a	historia
	e	aktie
	i	Hawaii
	o	tio
y	a	förnya
	e	nye
	o	embryo
ö	a	köa

Table 5.3: Legal final vowel sequences

when the performance of the system was evaluated.

5.1 The Rules

The rules that were created to detect errors by graphotactical means are looking at prefix and suffix of words, but not in the traditional grammatical sense of the words. A prefix and a suffix in this experiment is simply an initial and a final substring of a word. The length of the substrings is arbitrary.

A prefix rule contains an initial consonant, consonant cluster or vowel and a list of subsequent characters that creates an illegal string. For consonants and consonant clusters, this list can contain both consonants and vowels, while for vowels it only contains vowels. According to Sigurd (1965) a single initial vowel in Swedish can be followed by any consonant. A prefix rule written as a Perl regular expression might look like this:

```
/^vr[ouyöbcdfghjklmnpqrstvwxyz]+/
```

where the \wedge means beginning of the word, vr is the initial consonant cluster and the list of subsequent illegal characters is found between the square brackets. Any word that matches this regular expression, for example words beginning with vro or vrh , is output as a possible error.

A suffix rule consists mainly of the same parts as the prefix rules, a consonant, a consonant cluster or a vowel, a list of illegal following characters. As with the prefix rules, the list of illegal following characters can contain both consonants and vowels after consonants and consonant clusters, but only vowels after vowels. The suffix rule in addition takes care of the possibility that a word is followed by one or more punctuation marks. A suffix rule written as a Perl regular expression might look like this:

```
/jp[bcdfghjklmnpqrstvwxyz]+[\. \, \; \: \~ \)]*$/
```

where jp is the consonant cluster, the list of illegal subsequent characters is found between the first pair of square brackets, and the optional punctuation marks between the second pair of square brackets. The $\$$ marks word end. Any word that matches this regular expression, for example words ending with jpd or jpn is output as a possible error.

The rules that find alphanumeric combinations and words mixing upper and lower case letters are similar to the prefix and suffix rules. They too are formulated as Perl regular expressions. The rule for alphanumeric combinations:

```
$word =~ m/[a-zA-ZääöÄÄÖéè]+/ and $word =~ m/[0-9]/ and $word !~ /\-/
```

where the first matching succeeds if the word contain letters, upper or lower case, the second succeeds if the same word contain digits, and the third succeeds if the word does not contain a hyphen. This rule outputs strings that

contain both digits and letters but not a hyphen, which assures that common Swedish compounds like *70-talet* or *40-åringen* are not classified as possible errors.

The rule for strings mixing upper and lower case characters:

```
$ord =~ m/^[a-zääö]+/ and $ord =~ m/[A-ZÄÄÖ]+/
```

where the \wedge in the first regular expression means word beginning and then a lower case character from the list within square brackets must follow, the next regular expression checks if the word contains upper case letters. Any string matching the two regular expressions, i.e. first character lower case and an upper case character at any other position, is output as a possible error.

5.2 The Errors Found

Mostly non-words were output as possible errors: words with illegal prefixes,

```
bjektdefekt
hjälpstruktur
hchövs
kgenerslis
llir
shgularförm:
```

words with illegal suffixes,

```
anamnn
fördelakticj
kasz
modifikationp
schernatjsm
tolkninq
```

strings of letters containing special characters or punctuation marks in non-final position,

```
&kall
a~alys
häc,)er).
lexikonet,~vilket
pi~pr
samtliga.~bélägg
```

or strings containing at least four consonants in a row or strings containing no vowels,

```

bjbrk
cnsv
innchblrier
‘‘ordföijdsspråk’’,
rfnq’`
översälttningsuppgift.

```

Some correct words have been presented as possible errors because of their unusual spelling, like *determinism*, *algorithm*, *paradigm* and *bransch* that have unusual final consonant clusters, and *sfär* that has a very unusual initial consonant cluster. The words ending in *-ism* are a rather large group and would need to be fit into the graphotactical rules, while *algorithm* (together with *rytm*) and *sfär* (together with *sfinx*) rather should be allowed as exceptions in the system.

Other correct words presented as possible errors are compounds with an abbreviated part, like *mt-systemet* or *gpsg-inspirerad*. A rather large group of correct strings among the possible errors are the alphanumeric combinations. A string that mixes letters and digits are by definition a possible error to catch strings like **5**ubstitution (substitution) and **6**ver (över), but there are a lot of correct alphanumeric combinations for example in references, 1988:44ff, and enumerations, 1a, 1b, 7th etc.

5.3 Results

For the Swedish material two different tests have been made. First the large test corpus, 71 000 words, was scanned for errors and the output errors were categorised. Then the same test as in the statistical experiment was made on two hand corrected articles. The errors were counted before and after correction. In the graphotactical experiment no counting of errors generated in the correction procedure was made.

State-of-the art OCR systems give a result of up to 99% correct character recognition, which gives in average one error per 20 words (Kukich, 1992b). One error per 20 words would for my corpus give 3550 errors, and while the program found 2620 possible errors, about 370 of them were abbreviations (which could easily be filtered out), about 75 of them were correct non-compound words, and about 100 were acceptable alphanumeric combinations. This leaves about 1900 likely errors, which — if the error estimation holds! — would be a precision of around 75% at a recall of more than 50%.

Both the hand corrected articles contained less than 5% errors on word level which means that more than 99% of the characters have been correctly recognised (Kukich, 1992b). NODA83-03 contained 2200 words and 89 errors of which 24 were real word errors and 1 split error. (The words that got capitalised are counted among the real word errors.) The graphotactical rules presented 42 possible errors of which 19 were non-word errors, 1 was a correct Swedish word, 14 were correct foreign words (English) and 8 were abbreviations. NODA89-03 contained 1500 words and 65 errors, of which 3 were real word errors and 4 were split errors. The graphotactical rules presented 41 words as possible

errors of which 15 were non-word errors, 14 were correct Swedish words, 10 were correct foreign words (English) and 7 were abbreviations. This means that the graphotactical rules managed to find 30% of the non-word errors in NODA83-03 and 26% in NODA89-03, and that when the abbreviations have been filtered out only one and 10 correct Swedish words respectively have been presented as possible errors.

At the moment the program cannot deal with split errors, since it is using the space as word delimiter. If a split error results in one or more invalid strings the program will signal the strings as possible errors, but it is not possible to correct the split error by eliminating the space. The real word errors are for obvious reasons not dealt with either.

Chapter 6

Implementation

The implementation of the tests described above is a correction tool for Swedish text written in Perl/Tk. The input text should be in HTML format. The program detects errors with the graphotactical rules described above and enables the user to proofread a text in a non-linear way. All occurrences for each possible error are displayed to the user at the same time, with a small context. This gives the user the possibility to decide if all the occurrences really are to be corrected, or if one occurrence is correct (maybe an unusual abbreviation or an acronym). In a traditional spell checker the user probably would have clicked "Change all" and missed the single correct occurrence since it was not displayed at the time. It is also possible to give different corrections to different occurrences.

6.1 Technical Description

The program is divided in two files, one main program file and one separate Perl module, *final.pm*, that contains the longer subroutines. Putting the three longest subroutines in a module makes the main program more manageable and easier to read. The program is called with the program name and one parameter, the HTML file to be corrected:

```
perl finalprototyp.perl html-file
```

The program reads the input file line by line and does two things in parallel:

1. The text is put together line after line to one long string of text with spaces separating the words (henceforth called the concordance line). To avoid the program matching outside the file when it takes a context of 100 characters on each side of the actual word, a string of 150 characters is added in the beginning and at the end of the concordance line.
2. Every line is split in words and the words are stored in a hash table. The hash table becomes a list of all unique word types in which the search for error words will be made. The hash table structure makes searching very fast.

Figure 6.1: Graphical Interface

The main loop is closed by a single call to `errorfinder`, see description below, and the first call to `makeConcordance`, see description below.

6.2 Graphical Interface

A graphical interface is made with Perl/Tk: The title of the program window is set to *Nu rättas* and the name of the input file. The window contains an entry widget that displays the actual possible error word, a listbox that contains all occurrences of the possible error with context, an entry widget where the user enters the correction, a change-button marked with *ändra*, a next-button marked with *nästa*, and a quit-button marked with *avsluta*. See Fig 6.1.

6.3 Subroutines

6.3.1 errorfinder

`errorfinder` is one of the subroutines located in the separate module *final.pm* and is only called once by the program. `errorfinder` searches through the hash table of word types to find possible error words. To find the errors it uses regular expressions based on the Swedish graphotax. Every word is converted to lower case letters and run through all the regular expressions. Each time a word matches the regular expression, it is pushed onto an array (`@suspect`) of possible error words. The regular expressions are chained with `if, elsif, elsif...` to avoid that a word matches more than one regular expression and gets multiple occurrences in the array of possible errors.

6.3.2 makeConcordance

`makeConcordance` is called for every word in the array of possible errors. It shifts the first word from the array, and searches through the concordance line for every occurrence of the actual word. It makes a list of the occurrences with a context of 100 characters on each side of the word and makes a copy of the list. One of the lists will be displayed to the user, and the other will be used to match in the concordance line to make the changes. Then the list of occurrences that is to be displayed to the user is send to `fromhtml` (see description below) to get rid of the HTML tags and a smaller excerpt (35 characters on each side of the word) is extracted and displayed in the listbox.

6.3.3 fromhtml

`fromhtml` is one of the subroutines in the module. It removes HTML tags, < and >, and everything between them, and translates HTML code of the pattern &...; to their ASCII characters.

6.3.4 makeChanges

`makeChanges` gets the correction that the user has entered in the correction entry and sends it to `tohtml` (see description below) to replace special characters, if any, with &...; codes. The actual error word is then replaced with the correction at every occurrence in the concordance line. When replacing in the concordance line, the "long" occurrence is used, with 100 characters on each side of the error word and the HTML tags and codes intact. It empties the entry of the actual error word, the listbox and the correction entry, and makes a new call to `makeConcordance`.

6.3.5 tohtml

`tohtml` is one of the subroutines in the module. It replaces special characters with their &...; codes.

6.3.6 errorDialog

`errorDialog` displays a dialog box with the text *Du har inte angivit något rättningförslag* if the user clicks the change-button and no correction is entered in the correction entry.

6.3.7 changeDialog

`changeDialog` displays a dialog box that tells the user how many corrections that has been made.

6.3.8 naesta

`naesta` is called when the next-button is clicked, i.e when the user do not want to make any more corrections but go to next possible error word. It makes a new call to `makeConcordance` without making any changes and empties the entry of the actual error word, the correction entry and the listbox. The subroutine is completed by a new call to `makeConcordance`.

6.3.9 slut

`slut` is called when the user has clicked the quit-button and the dialog box `changeDialog` is displayed. It destroys `changeDialog` and ends the program.

6.3.10 sluta

`sluta` is called when the user has clicked the quit-button and the `changeDialog` is not displayed. It destroys the main window and ends the program.

When the `slut` or `sluta` subroutines are called, the concordance line is printed to standard out or to the specified output file with a new line character after every HTML tag.

6.4 Error Handling

There are two kinds of error handling: External error handling that tries to anticipate mistakes the user might do and react with a warning or an instruction to the user. The other kind — internal error handling — handles internal procedures for the program and sees to it that the different parts of the program gets the correct input parameters, and that the subroutines get the right input data. If this is not the case, the program gives alternative instructions to avoid a crash.

6.4.1 External Error Handling

When starting the program, the user is supposed to specify an input file. If no input file is specified, a message is printed to standard out: *Fel antal parametrar. Programmet anropas med en html-fil.* If the program cannot open the specified input file a message is printed to standard out: *file name går inte att öppna.* The program does not check if the input file really is an HTML file. If the user specifies a text file or a word file as input the program will try to correct it, but the result might be strange.

The user cannot delete words from the input file by changing them to the empty string. If the user clicks the change-button and no correction is entered in the correction entry a dialog box with the text *Du har inte angivit något rättningsförslag* is shown.

To avoid that the user clicks the change-button without marking any occurrences to be changed the first occurrence in the list displayed to the user is marked by default.

6.4.2 Internal Error Handling

To avoid that the program matches outside the text a string of 150 characters is added at the beginning and at the end of the concordance line. This means that it is possible to extract 100 characters on each side of the error word even if it is the first or the last in the input file.

The context of the error word that is to be displayed in the listbox is 35 characters on each side of the word. Before excerpting the 35 characters on each side of the error word, the program checks if it is possible. If there are less

than 35 characters on one side (or both sides) of the word, the program takes as many characters as it can find.

The subroutine `makeConcordance` always checks if there are possible error words left in the `@suspect` array. If there is not, the subroutine does not execute. Otherwise it would output a concordance of the empty string.

6.5 Possible Improvements

At the moment there is no possibility of undoing a made correction, since the program does not keep track of where the correction is made in the text. The program can not find the position in the text again, and thus can not undo the correction. Another consequence of this is that the user can do only one correction per occurrence.

It is not possible to go back to the previous error word and to see the concordance over that word again. The program does not keep the error words and can thus not go back and reconstruct the concordance.

The way the actual word is displayed could be better. For the moment it is done with an entry widget, and consequently the actual word is not centered over the occurrences. The ideal would be not to display the word separately, but to highlight it by color or other means in the concordance. This is not possible in the listbox widget.

When correcting an error the user should be able to change the scope of the error. If the program presents *översättn* as a possible error and the context looks like *översättn ing*, the user should be able to mark *översättn ing* as the error string and replace it with the correct string *översättning* without space. At the moment the user can not change the scope of the error string, thus split errors cannot be corrected even when observed by the user.

It is not possible to view the result in the same program as the corrections are made. At the moment the user has to open the corrected file in another program to see what effects the corrections had.

The main window of the GUI varies a lot in size, according to the length of the actual error word and it can sometimes get larger than the screen.

Chapter 7

Discussion

Optical scanning is a technique that will be widely used in the near future, as a method to move text resources from paper medium to electronic form. State-of-the-art OCR software still produces errors and post processing in the form of proofreading is often necessary. Proofreading optically scanned text by hand is difficult since many errors are caused by optical similarity between characters, which makes it hard to see the errors in the text. In addition, OCR is often used for very large text materials which makes the proofreading a time consuming task. To solve these problems it is necessary to develop good tools for proofreading, and they should be designed for proofreading of large text materials containing a lot of errors. This means that they should scan the text in a non-linear way, not give too many false alarms, show the error in a context and make it possible for the proofreader to give different corrections to different occurrences of the same error.

The graphotactical rules do not reach the same recall as a lexical error finding approach: since strings that do not violate the Swedish graphotax might still be non-words, for example *sermantik* (semantik) and *systemet* (systemet). Since the target material of this experiment is optically scanned text, often in large quantities, we are dealing with lots of text with lots of errors. The high recall of a lexical approach would also bring with it the large numbers of false alarms we have learnt to expect from these systems, and would, applied on optically scanned material, make the correction process impossible. The precision of this method although, will spare the user many of the false alarms and still clean up the text from a substantial part of the recognition errors.

The performance of the program could be improved both by improving the rules, by including the words ending in *-ism* and other sequences that the model does not cover yet, and by extending the program to look at medial character sequences. At this moment the model only checks initial and final sequences of words, which of course is a limitation made for this experiment. Sigurd discusses medial sequences too, but the time frame of this work has not allowed me to investigate these closer. The medial sequences would, however, be more complicated than the initial and final sequences, partly due to the fact that

Swedish shows an almost unlimited possibility of compounding, which can result in a large variety of character sequences at morpheme boundaries; partly because the distribution of the medial sequences is closely related to syllables and stress (Sigurd, 1965). Rules for the medial sequences could therefore not be based only on graphotax.

Chapter 8

References

- Allén, S. et al. 1975. *Nusvensk Frekvensordbok*. Stockholm.
- Allén, S. 1983. Inte bara idiom. In Proceedings of the 4th Nordic Conference on Computational Linguistics. Uppsala.
- Andersen, P. 1989. How Close Can We Get to the Ideal of Simple Transfer in Multi-lingual Machine Translation (MT)? In Proceedings of the 7th Nordic Conference on Computational Linguistics. Reykjavik.
- Bos, B., Bouma, G., Dings, J., Provoost, J. 1993. Natural Language Processing. Development of European Courses on Information and Datacom Engineering, DECIDE. University of Groningen. Groningen.
- Brodda, B. 1979. Något om de svenska ordens fonotax och morfotax: Iakttagelser med utgångspunkt från experiment med automatisk morfologisk analys. PILUS no. 38. Stockholm University.
- Dahlqvist, B. 1997. Kort om optisk inläsning av text. Uppsala universitet. Uppsala.
- Domeij, R., Hollman, J., Kann, V. 1994. Detection of spelling errors in Swedish not using a word list en clair. *Journal of Quantitative Linguistics*, 1:195-201.
- Ejerhed, E. & Bromley, H. 1985. A self-extending lexicon: description of a word learning program. In Proceedings of the 5th Nordic Conference on Computational Linguistics. Helsinki.
- Garlén, C. 1988. *Svenskans fonologi*. Studentlitteratur. Lund.
- Golding, A.R., & Schabes, Y. 1996. Combining Trigram-based and Feature-based methods for Context-Sensitive Spelling Correction. In Proceedings of the

34th Annual Meeting of the Association for Computational Linguistics. Santa Cruz, CA.

Grudin, J. 1983. Error patterns in skilled and novice transcription typing. In *Cognitive Aspects of Skilled Typewriting*. Ed. W.E. Cooper. New York.

Gustafsson, L.M. 1989. Händelsestyrd textgenerering. In Proceedings of the 7th Nordic Conference on Computational Linguistics. Reykjavik.

Hogan, C. 1999. OCR for Minority Languages. In Proceedings of the 1999 Symposium on Document Image Understanding Technology. Annapolis, Maryland.

Huisman, G. 1999. OCR Post Processing. University of Groningen. Groningen.

Kernigan, M. D., Gale, W. A., Church, K. W. 1990. A Spelling Correction Program Based on a Noisy Channel Model. In Proceedings of COLING 1990. Helsinki.

Kukich, K. 1992. Spelling Correction for the Telecommunications Network for the Deaf. In ACM Computing Surveys, Vol. 35, No. 5, 80-90.

Kukich, K. 1992. Techniques for Automatically Correcting Words in Text. In ACM Computing Surveys, Vol. 24, No. 4, 377-439.

Källgren, G. 1990. "The first million is hardest to get": Building a Large Tagged Corpus as Automatically as Possible. In Proceedings of COLING 90, Helsinki.

Lee, G., Lee, J-H. & Yoo, J. 1997. Multi-level post processing for Korean character recognition using morphological analysis and linguistic evaluation. Pattern Recognition 30(8): 1347 - 1360.

Meknavin, S., Kijirikul, B., Chotimonkol, A. Nuttee, C. 1998. Combining Trigram and Winnow in Thai OCR Error Correction. In Proceedings of COLING 1998. Montréal.

Sigurd, B. 1965. *Phonotactic Structures in Swedish*. Lund University. Lund.

Svenska Akademiens ordlista över svenska språket. 1999. Stockholm.

Takahashi, H., Itoh, N., Amano, T. & Yamashita, A. 1990. A Spelling Correction Method and Its Application to an OCR System. Pattern Recognition vol 23 3/4.

Tillenius, M. 1996. Efficient generation and ranking of spelling error corrections. NADA report TRITA-NA-E9621.

Tong, X. & Evans, D. 1996. A Statistical Approach to Automatic OCR Error Correction in Context. 4th Workshop on Very Large Corpora. Copenhagen.