Syllabification of Middle Dutch

Gosse Bouma
Rijksuniversiteit Groningen
g.bouma@rug.nl

Ben Hermans
Meertens Institute
ben.hermans@meertens.knaw.nl

Abstract

The study of spelling variation can be seen as a window allowing us to understand the phonological systems of the dialects of Middle Dutch, and to what extent they differed. Syllabic information is of great help in the study of spelling variation, but manual annotation of large corpora is a labor-intensive task. We present a method for automatic syllabification of words in Middle Dutch texts. We adapt an existing method for hyphenating (Modern) Dutch words by modifying the definition of nucleus and onset, and by adding a number of rules for dealing with spelling variation. The method combines a rule-based finite-state component and data-driven error-correction rules. The hyphenation accuracy of the system is 98.4% and word accuracy is 97.4%. We apply the method to a Middle Dutch corpus and show that the resulting annotation allows us to study temporal and regional variation in phonology as reflected in spelling.

1 Introduction

The Corpus Van Reenen Mulder[1] (van Reenen and Mulder, 1993; Coussé, 2010) consists of Dutch legal texts from the 14th Century. It is one of the very few resources from that period that has been made available in electronic form. Due to its size and composition (over 2.700 documents dated between 1300 and 1399 and from all Dutch speaking regions in the Netherlands and Flanders) it is ideally suited for the study of spelling variation.

Our research is motivated by the need to study spelling variation of older Dutch texts. To a large extent spelling variation is determined by phonological systematicity; that is, differences in spelling can be a consequence of differences between dialects. Take for instance the difference between <*priester*>, <*preyster*> and <*prester*> *'priest'*. It is possible that the differences in the spelling of the vowel in the first syllable reflects a difference in the phonological status of this vowel in the respective dialects; a centralizing diphtong in the case of <*priester*>, a falling diphthong in the case of <*preyster*>, and a long mid vowel in the case of <*prester*>. This is a legitimate analysis, because in the Dutch dialect area as

[1] www.diachronie.nl/corpora/crm14

it is now, these differences are still attested. It is highly important to find cases where orthographic variation is determined by phonological systematicity, because they allow us to gain more insight into the phonological structure of the dialects of Middle Dutch. It would be the first step in the construction of a dialect atlas of the dialects of Middle Dutch.

To realize this goal the presence of information regarding syllable structure is of great help. With the presence of syllabic information it becomes a lot easier to determine the class of graphemes that are allowed in a specific position in a syllable, like the coda. That, again, might help us to gain more insight into the phonological processes that played a role in the domain of the coda. Consider, for instance, the fact that the grapheme <*gh*> is found fairly often in the coda position. Examples are <*meghtich*> *'mightly'* and <*ghetyghnesse*> *'testimony'*. Normally, the grapheme <*gh*> represents the voiced velar fricative. What does the presence of this grapheme in coda position tell us? It suggests perhaps that, in some Middle Dutch dialects, the process of devoicing did not apply in the same way it applies in the modern dialects. The study of the distribution of the graphemes representing voiced obstruents might reveal significant information regarding the processes of devoicing, voice assimilation and the difference between dialects of Middle Dutch with respect to these processes.

A second example of the importance of syllable structure is the distribution of <*l*> in the coda. An important difference between dialects is to what extent they allow the grapheme <*l*> in this position. In some dialects we find forms like <*arnolde*> (proper name) and <*goldene*> *'golden'*, whereas in other dialects we find the corresponding <*arnoude*> and <*goudene*>. The study of the distribution of <*l*> in coda position might reveal how exactly the sound change whereby */l/* vocalized to a glide developed over time, how it spread regionally, and in which phonological environments it took place.

In short, the study of spelling variation can be seen as a window allowing us to understand the phonological systems of the dialects of Middle Dutch, and to what extent they differed. By extenstion it allows us to study the development of phonological processes over time, and the way they spread regionally. The information of syllabic information is of great help in the study of spelling variation, and this is the reason why we think it is important to construct a method for automatic syllabification of words in Middle Dutch texts.

In this paper, we present a method to automatically add syllable boundaries to Middle Dutch words. The method adapts an existing method for hyphenating Modern Dutch words to Middle Dutch and consists of two parts: a finite-state transducer which implements the two main rules for Dutch hyphenation, and a statistical component that automatically learns rules to correct errors in the output of the first method.

For testing and evaluation as well as for automatically learning error-correction rules, we created a gold standard list of hyphenated word types. To this end, 50% of the word types from the corpus was hyphenated automatically using the rule-based finite-state system. The output of the automatic system was manually corrected

to obtain a gold standard. Note that given the limited size of the corpus (approximately 650.000 tokens and 43.000 types),[2] we could also have corrected all types and used the corrected list to hyphenate running text. The advantage of our automatic method over a dictionary-based method, however, is that it will also be able to hyphenate words that are not in the dictionary, and thus, its coverage on unseen corpus-data will probably be better than a method based only on dictionary look-up. In practice, highest accuracy is probably obtained by applying dictionary look-up for known words, and the automatic method for unknown words.

In the context of two recent projects (Adelheid[3], InPolder[4]) the texts in the corpus van Reenen Mulder have been annotated with lemmas (in Modern Dutch spelling) and morphological structure. Syllable boundaries, however, are not indicated. As there is a considerable gap between the original spelling and the corresponding lemma's in Modern Dutch, it is not easy to determine syllable boundaries or hyphenation points on the basis of a hyphenated Modern Dutch dictionary.

Below, we describe previous work on hyphenating (Modern) Dutch and relevant differences between Modern and Middle Dutch. Next, we present our implementation of the finite-state hyphenation method, which achieves a hyphenation accuracy of 94.0%. In section 5, we apply transformation-based learning and improve accuracy to 98.4%. In section 6, we give some examples of using the corpus for studying temporal and regional tendencies in spelling variation.

2 Hyphenating Modern Dutch

Bouma (2003) describes a method for accurate hyphenation of Modern Dutch text. It consists of two steps: a finite-state transducer that implements the maximum onset principle, the most important rule for hyphenating Dutch words, and a transformation-based learning component that, given a word list of correctly hyphenated words, automatically learns rules to correct errors produced by the finite-state transducer. The system is trained and evaluated on hyphenated word forms obtained from CELEX. The reported hyphenation accuracy (i.e. percentage of correctly inserted hyphens) is 99.3% and the word accuracy (i.e. percentage of correctly hyphenated words) is 98.2%.

Below, we give an informal overview of the system, emphasizing those aspects that will need reconsideration for our current task.

The rules for syllabifying words in Dutch follow two general principles:

1. Syllable boundaries cannot cross morpheme boundaries.

2. The maximum onset prinpiciple is respected. That is, consonants that may be added to either a preceding or following syllable are added to the onset of

[2] This is an approximation, as the transcription contains diacritic tokens to indicate unclear parts of the original manuscripts, words written as one, etc. We ignored those in our counts.

[3] http://adelheid.ruhosting.nl

[4] http://depot.knaw.nl/8914/

the following syllable if this does not violate constraints on onset clusters.

The finite-state method for hyphenating words only implements the second constraint. The reason for ignoring the first constraint is that detecting morpheme boundaries is hard, and requires, at least, a detailed lexicon and an implementation of morphological rules. The second constraint can be implemented using finite-state techniques. A detailed description is in Bouma (2003). That solution involves the following steps:

1. Mark the beginning of a word (represented as a sequence of characters).
2. Mark the beginning and end of each nucleus in a word.
3. Insert a hyphen at each position between a nucleus and a following nucleus, in such a way that the onset of the second nucleus is maximal.
4. Remove all markers except the hyphens.

Each of these steps can be implemented as a finite-state transducer, and the resulting system is then the composition of these transducers. The implementation of the rules is greatly simplified by the *replace* operator (Karttunen, 1995; Gerdemann and van Noord, 1999), a finite-state method for implementing phonological rules (i.e. contextually sensitive rules for replacing one symbol sequence by another).

In step 2, the method requires that each *nucleus* is marked. A nucleus is defined here as the maximal sequence of characters (going from left to right) that can represent a vowel or diphtong in Dutch. To this end, a listing of all possible nucleus character sequences is provided. Similarly, in step 3, the method requires that the onset of the second syllable follows the rules of Dutch orthography. Again, this is implemented by providing a list of all possible onset character strings. No constraints are imposed on the coda of a syllable, other than that it must consist of a sequence of consonants. An example of this algorithm for the word <*aardappel*> *'potato'*, a compound of <*aard*> *'earth'* and <*appel*> *'apple'* is shown below:

```
aardappel
    ⇓
+aardappel
    ⇓
+@aa@rd@a@pp@e@l
    ⇓
+@aa@r-d@a@p-p@e@l
    ⇓
aar-dap-pel
```

Note that the sequence <*aa*> is marked as a single nucleus, in spite of the fact that a letter <*a*> can also form a nucleus by itself. The output of the system contains an error, as it identifies `aar-dap` as a syllable boundary, where this should

have been `aard-ap`, following the morphological boundary between <*aard*> and <*appel*>.

The hyphenation accuracy of a system that only uses the notions nucleus and onset, is 94.5%, and word accuracy is 86.1%. To improve accuracy, one can use data-driven machine learning techniques that learn from correctly segmented words. In Bouma (2003), a method is presented that uses transformation-based learning (Brill, 1995; Ngai and Florian, 2001). Given a word, the system considers both the hyphenation predicted by the finite-state system and the correct hyphenation produced by a human expert. By inspecting large data samples, the system learns rules that correct frequent errors in the system output. In Bouma (2003), 290.000 hyphenated words from CELEX are used for training, and a hyphenation accuracy of 99.3% and a word accuracy of 98.2% is achieved. This accuracy is comparable to that of state-of-the-art hyphenation methods, such as the hyphenation patterns implemented in the text typesetting package LaTeX. An interesting feature of transformation-based learning is that the error-correcting rules can themselves be interpreted as finite-state transducers, and thus can be composed with the baseline finite-state hyphenator to obtain a highly efficient and accurate finite-state hyphenator.

3 Challenges

Adapting the method described above to Middle Dutch requires that we modify the definition of nucleus and onset and deal with some peculiarities of Middle Dutch spelling.

Middle Dutch texts exhibit a substantial amount of spelling variation, as illustrated in table 1. As a consequence, nuclei and onsets will also exhibit a wider range of variation than in Modern Dutch. The nucleus <*ey*> in <*borghermeyster*>, for instance, does not exist in Modern Dutch. More in general, the spelling of long vowels sometimes involves doubling of the character (as in Modern Dutch), but in other cases addition of <*e*>, <*i*> or <*y*> Thus, we find <*aan*>, <*aen*>, <*ain*>, and <*ayn*>.[5]

While spelling variation in itself does not make the hyphenation task harder (it just requires adding alternative spellings of nuclei and onsets), there are some patterns that do require special attention:[6]

- The characters <*i*> and <*j*> are interchanged frequently (<*iaer*> vs <*jaer*> *'year'*, <*ighelic*> vs. <*jghelic*> *'in fact'*).
- Similarly, <*u*> and <*v*> are often used interchangeably: <*uerclaringhen*>

[5] Van Halteren et al., `www.ccl.kuleuven.be/CLARIN/vanhalteren.pdf`, and Kestemont et al. (2010) address the issue of spelling normalization, where tokens in the original text are linked to the most likely lemma in Modern Dutch.

[6] These are all a consequence of the fact that Middle Dutch orthography is influenced by Latin, which did not distinguish between <*i*> and <*j*>, <*v*> and <*u*>, and did not have a <*w*>.

borgchermestere	burgeremeystere	burghermeisters
borgermestere	burgermeesteren	burghermestere
borghemeestere	burgermeister	burghermeysters
borghemeyster	burgermeystere	burghmeester
borghermeester	burghemeesteren	burghmeisters
borghermeistere	burghemeisteren	burghmesters
borghermester	burghemeysteren	burghmeysteren
borghermeyster	burghemeysters	burghmeysters
borhermestere	burghermeestere	burgmesters
burchmeester	burghermeistere	

Table 1: Spelling variants of the Dutch word for *'mayor'*. Inflected forms are only included if the uninflected form with the same spelling was absent.

vs. <*verclaringhen*> *'statement'*, <*uerstaen*> vs. <*verstaen*> *'understand'*, <*zeuentien*> vs. <*zeventien*> *'seventeen'*.

- The letter <*w*> is often used for the diphtong <*uu*> (<*uutghesproken*> vs. <*wtghesproken*> *'stated'*, <*zuutzide*> vs. <*zwtzide*> *'southside'*).
- Finally, double <*v*> is sometimes used to denote a vowel (long <*u*>) (as in <*hvvs*> (*'house'*)) or a consonant (<*w*>) as in <*gesvvoren*> (*'sworn'*).

This makes syllabification hard, as <*i*>, which only is (part of) a nucleus in modern spelling, may also be (part of) an onset in Middle Dutch. Similarly, <*j*> is only used in onsets in Modern Dutch (except for the diphtong <*ij*>), but can also be (part of) a nucleus in Middle Dutch. The same ambiguity holds for the character pairs <*u*> and <*v*>, the character <*w*> and the character bigram <*vv*>.

4 Adapting a Rule-based Hyphenator

As the general principles of syllabification have not changed, the general architecture of the rule-based, finite-state, hyphenator for Modern Dutch outlined in section 2 can remain unchanged. To account for the difference in orthography, however, we must adapt the definition of nucleus and onset. Furthermore, characters `i`, `j`, `u`, `v`, and `w` require special attention.

4.1 Spelling issues

To make the task of identifying nuclei and onsets more accurate, we first replace the character `u` with `U` in those cases where `u` functions as consonant, and replace `v`, `w`, and `j` with `V`, `W` and `J` respectively, in contexts where these function as vowels:

nucleus a, aaC, ae, ai, au, e, ee, ei, eu, ey, i, ie, ii, iae, J, o, oe, ooC, ou, oi, oy, u, uuC, ue, uy, ui, V, Vy, W, y, ye

onset b, bl, br, c, ch, cl, cr, d, dr, dw, f, fl, fr, g, gh, gl, gr, h, j, k, kl, kn, kr, l, m, n, p, ph, pl, pr, Q, r, s, sc, sch, schr, scr, sl, sn, sp, spl, spr, st, str, t, th, tj, tr, U, v, vl, vr, w, wr, x, z, zw

Table 2: A listing of the definition of nucleus and onset. Upper case C denotes any consonant. Upper case Q denotes the letter combination qu. Upper case J,U,V,W indicate occurrences of lower case j,u,v,w that function as vowel (<*j*>,<*v*>,<*w*> or consonant<*u*>).

- In the sequences aue, eue, and oui, u almost always functions as a v. Therefore, we replace such sequences with aUe, eUe, and oUi, respectively, where we use U as the character that denotes a u functioning as a consonant.
- In the sequences C_1vC_2 and $+vC_2$, v almost always functions as u. (We impose some restrictions on C_2 to prevent over-generalization.) We replace the v in such sequences with V to denote a v functioning as a vowel character.
- In the sequences C_1wC_2 and $+wC_2$, w almost always functions as uu. We replace the w in such sequences with W to denote a w functioning as a vowel character.
- In the sequences $+jn$ and $+jm$, j functions as i. Therefore, we replace j with J to denote a j functioning as vowel.

After these rules have been applied, the steps that identify nuclei and syllable boundaries are applied as in the finite-state method presented above.

4.2 Updating definitions

A nucleus is represented orthographically as a sequence of one, two, or sometimes three vowel characters. Middle Dutch spelling allows for some sequences that are not used in Modern Dutch (i.e. <*ae*>, <*ai*>, <*ey*>, <*ii*>, <*iae*>, <*oy*>, <*uy*>, <*ye*>). We inspected the word list to find the most frequent cases and incorporated these in the definition of nucleus, as shown in table 2. The spelling of onsets is to a large extent identical to that in Modern Dutch.

The nuclei aaC, ooC and uuC in the list are not quite in accordance with linguistic notions, but were introduced to make the prediction of hyphenation points more accurate. In Modern Dutch, long vowels in closed syllables are represented by a double character <*aa*>, <*oo*> or <*uu*>.[7] Therefore, if we encounter such

[7]The long vowel <*ee*> is an exception, as this can also occur in word-final open syllables, i.e. <*zee*> *'sea'*.

a sequence, we know the syllable of which it is the nucleus has to contain a non-empty coda. By marking the following consonant as part of the nucleus in such cases, we prevent the maximum onset principle from considering the consonant as part of the onset of a following syllable. The problem can be illustrated with a word like <*clooster*>, *'monastry'*. By recognizing a nucleus `oos`, we predict the hyphenation `cloos-ter`. Had we predicted the nucleus `oo`, we would obtain the hyphenation `cloo-ster`, as <*st*> is a possible onset. It should be noted, though, that the rule that says that <*aa*>, <*oo*>, and <*uu*> are always followed by a non-empty coda is not absolute (in contrast with Modern Dutch). So, we do also find examples such as <*coo-pen*> and <*boo-de*> and thus this strategy is less accurate in Middle Dutch than in Modern Dutch.

4.3 Implementation

We implemented the hyphenation system as a sequence of finite-state transducers. The steps in the algorithm are the same as for Modern Dutch, except that we introduce one additional step where certain letters are replaced by upper case letters in order to make the following steps more accurate.

1. Mark the beginning of a word (represented as a sequence of characters).
2. Replace letters `j,u,v,w` and bigram `qu` with an upper case letter in certain contexts.
3. Mark the beginning and end of each nucleus in a word.
4. Insert a hyphen at each position between a nucleus and a following nucleus, in such a way that the onset of the second nucleus is maximal.
5. Remove all markers except hyphens, and convert upper case letters to their lower case counterparts.

The most crucial parts are steps 3 and 4, which require a definition of nucleus and onset. We used the definitions in figure 2. We arrived at these definitions by iterative testing on the list of types from the corpus.

The manually developed rule-based system is useful mostly as a base-line system that helped us in creating a manually corrected list of word types with hyphenation points. That is, we compiled a word list from our corpus and automatically hyphenated the word types using our finite-state hyphenator. 50% of the resulting data was corrected by a human expert. As the accuracy of the rule-based system is relatively high, the amount of manual labor was modest and far more efficient than an approach where hyphenation patterns have to be added manually to all words.

After creation of the gold standard word list, we observed that the automatic system in fact achieved 94.0% hyphenation accuracy and 90.1% word accuracy.

word	(*answered*)	a n d w e r d e
system	*an-dwer-de*	0 0 1 0 0 0 1 0
correct	*and-wer-de*	0 0 2 0 0 0 1 0
word	(*pilgrimage*)	b e d e v a e r d
system	*be-deu-aerd*	0 0 1 0 0 1 0 0 0
correct	*be-de-uaerd*	0 0 1 0 0 9 0 0 0

Table 3: Aligning system output and correct hyphenation patterns

5 Applying Transformation-based learning

By comparing the output of the automatic finite state hyphenation program with the correct hyphenation in our segmented word list, we can detect where the program makes errors. If we can find patterns or regularities in the errors, we can try to add rules to the hyphenator that would correct or prevent these errors.

Transformation-based learning Brill (1995); Ngai and Florian (2001) is a machine learning method that automatically tries to find the rule that corrects most errors in the data. The score of a rule (on a set of training data) is the number of correctly corrected errors minus the number of newly introduced errors (as the rule will usually also apply to a number of cases that were actually correct). The method has been used mostly in part-of-speech tagging, and requires annotated data for training. Given a corpus that is both manually annotated (providing the gold standard) and annotated by a baseline POS tagger (for instance, a system that always assigns the most frequent POS for a word), TBL learns rules that replace an incorrect POS-tag in the system output by a correct POS-tag. After applying the rule with the highest score to the output of the baseline system, rule scores are recomputed and the rule with the highest score on the modified data is applied, and so on until a stopping criterion is reached.

Correcting hyphenation errors can be seen as a similar task. We first assign to each letter in an input word either the value 0 or 1, where 0 stands for 'not preceded by a hyphen' and 1 stands for 'preceded by a hyphen'. Furthermore, the gold-standard hyphenation pattern is aligned with the system output by using two more codes, 2 in case the hyphen actually has to be placed one more position to the right, and 9 in case a hyphen has to be placed one position to the left. As explained in Bouma (2003) this coding has the advantage that the correction of a hyphenation mistake in general requires 1 instead of 2 correction rules. Two examples of this alignment scheme are given in table 3.

We performed 10-fold cross-validation, where in each experiment the system is trained on 90% of the data, and test on the remaining 10%. The highest scoring rules for one of the experiments are given in table 4.

After applying the rules to a held out portion of the data, we obtain an average hyphenation accuracy of 97.90% (s.d. = 0.173) and an average word accuracy of

-ster	→	s-ter	eu-a	→	e-ua
-ru	→	r-u	+io-	→	+io
-fl	→	f-l	c+hei	→	ch-ei
-y	→	y	l-ue	→	lu-e

Table 4: The highest scoring error correcting rules learned by TBL

96.5% (s.d. = 0.317).

Roche and Schabes (1997) have shown that the rules learned by TBL can be interpreted as finite-state transducers that replace a symbol by another symbol in a given context. Sequential rule application corresponds to the composition of the transducers for the individual rules. An end-to-end system, finally, that hyphenates words, can be obtained by the composition of the finite-state transducer that represents the rule-based system and the finite-state transducer that implements all TBL rules. Using the FSA tools, we have computed this finite-state transducer (131 states, executable is 442 kB.)

6 Preliminary Exploration of results

The segmentation program that is the result of combining the rule-based and error-driven components can be applied to all of the texts in the Corpus van Reenen Mulder (CRM14). The result is texts in which each word has been segmented into syllables with an accuracy of over 97%.[8]

The metadata in CRM14 provides both the year and location of the manuscripts. Thus, we can easily compute the relative frequency of syllables, onsets, or nuclei over a period of 100 years (1300-1400). For instance, onsets <*gh*> and <*g*> are often used interchangeably, as witnessed by the fact that we find over 3000 minimal pairs, i.e. word types that differ only in the choice for <*gh*> vs. <*g*> such as <*al-ghe-heel*> vs. <*al-ge-heel*>. Similarly, the nuclei <*ei*> and <*ey*> are often used interchangably (over 700 minimal pairs).

Figure 1 compares the change in relative frequency of the onsets <*gh*> and <*g*> over time. It shows that during the period represented in the corpus, the frequency of onset <*gh*> is falling while the frequency of onset <*g*> rises. Note that for this comparison, obtaining accurate frequency estimates from a corpus with unsegmented words is challenging, as determining whether *g* is part of a coda or onset in word-internal positions is hard without actually segmenting the word.

We can also study regional tendencies in spelling. The geographical distribution of <*ei*> as a percentage of all <*ei*> + <*ey*> occurrences for a given location,[9]

[8]The word accuracy on running text is probably higher than that for a word list for the same text. One reason is the fact that monosyllabic words tend to be frequent. As the programme hardly makes mistakes on monosyllabic words (as identification of a nucleus is close to perfect), the overall accuracy on running text will go up.

[9]Only shown for locations for which at least 10 occurrences of both forms were available.

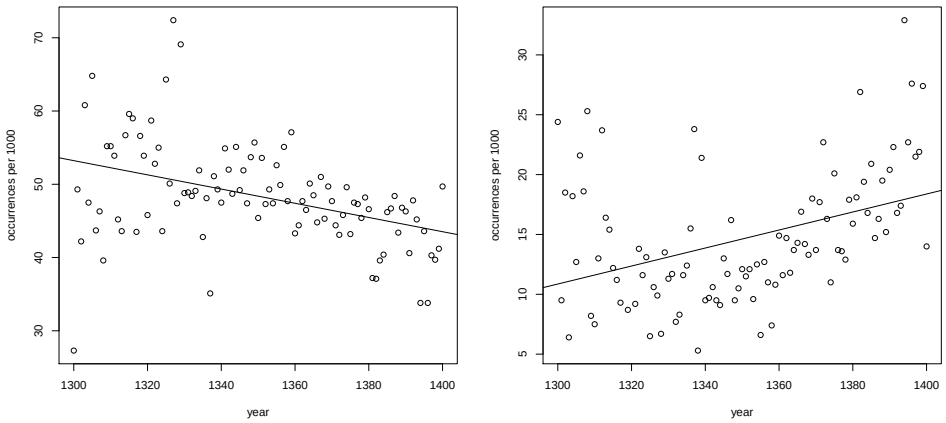
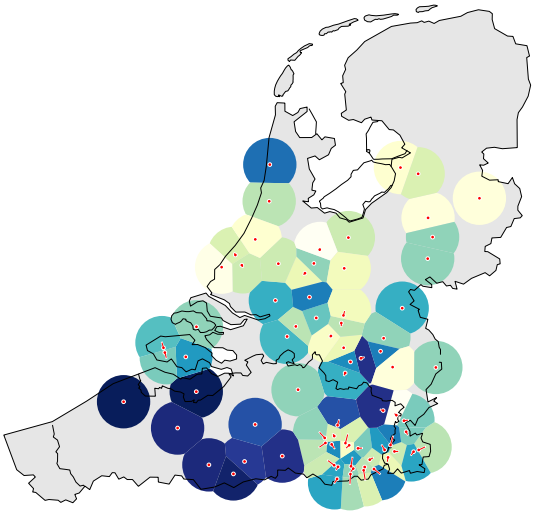


Figure 1: Distribution of the onsets *gh* (left) and *g* (right) over time (1300-1400).

Figure 2: <*ei*> occurrences as a percentage of the total number of <*ei*> and <*ey*> occurrences for various locations. Darker colors indicate a higher percentage of <*ei*>.



is given in Figure 2. It shows that preference for <*ei*> was stronger in the south of the Netherlands than in the north.[10]

7 Conclusion

The study of phonological processes in historical text can benefit from accurate information about the morphological and phonological structure of words. We have presented a method for accurate syllabification of Middle Dutch texts, using a finite-state and data-driven method originally developed for Modern Dutch. The result can be compiled into an efficient transducer that can be used to automatically

[10]The map was created using the Gapmap software, `www.gabmap.nl`.

annotate large corpora from the given era with syllable boundaries. An obvious candidate is the Corpus Gysseling.[11]

By applying this method to the complete CRM14, we obtain a corpus annotated with syllable boundaries. We demonstrate that this information can be used to study both temporal and regional variation in the distribution of onsets and nuclei. In future work, we hope to show that this can be the basis of deeper and more principled studies into the phonology of Middle Dutch.

References

Gosse Bouma. Finite state methods for hyphenation. *Journal of Natural Language Engineering*, 9:5–20, 2003. Special Issue on Finite State Methods in NLP.

Eric Brill. Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computational Linguistics*, 21: 543–566, 1995.

Evie Coussé. Een digitaal compilatiecorpus historisch Nederlands. *Lexikos*, 20: 123–142, 2010.

Dale Gerdemann and Gertjan van Noord. Transducers from rewrite rules with backreferences. In *Proceedings of the Ninth Conference of the European Chapter of the Association for Computational Linguistics*, pages 126–133, Bergen, 1999.

Lauri Karttunen. The replace operator. In *33th Annual Meeting of the Association for Computational Linguistics*, pages 16–23, Boston, Massachusetts, 1995.

M. Kestemont, W. Daelemans, and G. De Pauw. Weigh your words-memory-based lemmatization for Middle Dutch. *Literary and Linguistic Computing*, 25(3): 287–301, 2010.

Grace Ngai and Radu Florian. Transformation-based learning in the fast lane. In *Proceedings of the second conference of the North American chapter of the ACL*, pages 40–47, Pittsburgh, 2001.

Emmanuel Roche and Yves Schabes. Deterministic part-of-speech tagging with finite-state transducers. In Emmanuel Roche and Yves Schabes, editors, *Finite state language processing*, pages 205–239. MIT Press, Cambridge, Mass., 1997.

P. T. van Reenen and M. Mulder. Een gegevensbank van 14de-eeuwse Middelnederlandse dialecten op de computer. *Lexikos*, 3:259–281, 1993.

[11] `http://gysseling.corpus.taalbanknederlands.inl.nl/cqlwebapp/search.html`