CHAPTER FOUR

GRAPHEME-TO-PHONEME CONVERSION

4.1 INTRODUCTION

In this chapter we analyse the grapheme-to-phoneme (g-to-p) conversion task through a number of experiments. Our aim is obtain a pronunciation modelling mechanism that is well suited to bootstrapping. We choose an instance based learning approach, with Dynamically Expanding Context (DEC) as the baseline algorithm, for reasons discussed in Section 4.2. We utilise the pronunciation dictionaries described in Section 4.3 to analyse various aspects of the task, and to benchmark our results. As DEC is sensitive to alignment errors, we first analyse grapheme-to-phoneme alignment accuracy (in Section 4.4), and define the alignment approach we utilise in subsequent experiments. We then proceed to analyse a number of variations of DEC, and suggest small adaptations to the standard algorithm (Section 4.5). These variations lead to the definition of a new grapheme-to-phoneme conversion algorithm described in Section 4.6. This algorithm – *Default & Refine* – has a number of attractive properties that makes it suitable for bootstrapping.

4.2 BASELINE ALGORITHM

As discussed in Section 3.4.1, the ideal grapheme-to-phoneme conversion mechanism will have the following characteristics:

1. High predictive ability, even for very small training set sizes.
2. Exact representation of training data.
3. Low computational cost (both for rule extraction and pronunciation prediction).
4. Robustness to noise in the training data.

Of the approaches discussed in Section 2.2.2, we exclude any that require linguistic input (such as finite state transduction) or extensive computational resources (such as meta-classifiers). Of the remaining approaches, most exhibit comparable asymptotic performance, with the best results currently achieved by pronunciation by analogy (PbA) approaches and instance-based learning methods, as described earlier.

Both PbA approaches and instance-based learning methods can provide exact representation after conversion, as required. Also, the computational complexity of examples within both of these classes of algorithms are within acceptable limits, with PbA approaches providing some advantage with regard to computational cost [25]. As bootstrapping is typically not the aim of grapheme-to-phoneme approaches, little information is available with regard to robustness to noise. The first requirement then becomes the deciding factor for choice of algorithm: how well does the algorithm generalise from very small data sets. Again, explicit information is not available, but it seems from the results provided by Damper *et al* in [25] that the PbA algorithm only starts to generalise well when the training dictionary is of sufficient size[1]. We therefore choose an example of instance-based learning as the basis for our initial experimentation. Specifically, we choose Dynamically Expanding Context (DEC), an algorithm that is simple to implement, and generalises fairly well from a small training set.

4.3 EXPERIMENTAL DATA AND APPROACH

We utilise the following databases during experiments:

- *NETtalk*, a publicly available 20,008-word English pronunciation dictionary [20], derived from Miriam Webster's pocket dictionary (1974). Hand-crafted grapheme-to-phoneme alignments are included in the electronic version.
- *FONILEX*, a publicly available pronunciation dictionary of Dutch words as spoken in the Flemish part of Belgium [19]. We obtained the exact 173,873-word pre-aligned version of the dictionary as used by Hoste [41].
- *OALD*, a publicly available English pronunciation dictionary [18]. We obtained the exact 60,399-word pre-aligned version of the dictionary as used by Black [23].
- *Afrikaans A*, a 5,013-word Afrikaans pronunciation dictionary, built using the bootstrapping system and developed as part of this thesis. This dictionary was transcribed by a linguistically sophisticated first-language Afrikaans speaker and manually verified by the author. Of the 5,013 words, 90 words are invalid: the remaining 4,923 words are all valid and distinct.
- *Afrikaans B*, a 8,053-word Afrikaans pronunciation dictionary, built using the bootstrapping system and developed as part of this thesis. This dictionary was bootstrapped from *Afrikaans*

[1] When trained on the (American English) Teachers' Word Book (TWB), the PbA algorithm that was evaluated achieved approximately 40% word accuracy after 2000 words [25]

A and transcribed by a linguistically sophisticated first-language Afrikaans speaker, but not exhaustively verified. (Some verification was performed, as described in Section 6.5.) Of the 8,053 words, 271 words are invalid: the remaining 7,782 words are all valid and distinct.

Where any of the above databases include pronunciation variants (one word associated with two or more valid pronunciations), all but the first pronunciation variant are removed from the database, prior to dividing the database into training and test sets. When we report on results, we use the term *phoneme correctness* to specify the percentage of phonemes identified correctly, *phoneme accuracy* as the number of correct phonemes minus the number of insertions, divided by the total number of phonemes in a correct pronunciation, and *word accuracy* to specify the percentage of words completely correct. While we typically report on phoneme accuracy only, phoneme correctness is sometimes included in order to provide a comparative measure with results from other sources. Unless otherwise stated, we perform 10-fold cross-validation. During 10-fold cross-validation we subdivide the entire corpus randomly into 10 distinct sub-sections, and then perform 10 training/testing experiments, training on nine of the sub-sections and testing on the tenth. For the different measurements (word accuracy, phoneme accuracy, phoneme correctness) we report on the standard deviation of the mean of each of these measurements, indicated by ${\sigma_{10}}^2$. Where there is uncertainty with regard to the measure used in a benchmark result, word accuracy provides the least ambiguous comparison.

As in previous sections, we use the format

$$(x_1..x_m, g, y_1..y_n) \rightarrow p \tag{4.1}$$

to specify extracted grapheme-to-phoneme rules. Here g indicates the focal grapheme, x_i and y_j the graphemic context, and p the phonemic realisation of the grapheme g. We also use a more compact representation:

$$x_1..x_m - g - y_1..y_n \rightarrow p \tag{4.2}$$

to indicate the same rule. Note that each grapheme specifies a separate element, even though these separate elements are written next to each other (without spaces or other indicators of element boundary.)

4.4 GRAPHEME-TO-PHONEME ALIGNMENT

Errors in grapheme-to-phoneme alignment do not affect different rule extraction techniques to the same extent. DEC-based rule extraction mechanisms are sensitive to alignment accuracy. For example, the correct DEC extraction rule for the grapheme-pair 'ee' in English is $-e-e \rightarrow iy$ and $e-e- \rightarrow \phi$ where ϕ indicates the null phoneme. If the system incorrectly aligns the words "keen"

[2] If the mean of a random variable is estimated with n independent measurements, and the standard deviation of those measurements is σ, the standard deviation of the mean is $\sigma_n = \frac{\sigma}{\sqrt{n}}$.

and "seen" as follows: $k\ e\ e\ n \rightarrow k\ iy\ \phi\ n$ and $s\ e\ e\ n \rightarrow s\ \phi\ iy\ n$, DEC will not be able to extract the fairly simple rule specified above, as the two words provide conflicting evidence with regard to the pronunciation of the grapheme pair 'ee'. Note that the linguistic accuracy of the position of the null phoneme is not important, as long as the choice of position is consistent across the set of training instances. As DEC is sensitive to alignment accuracy, we optimise the grapheme-to-phoneme alignment process before analysing the grapheme-to-phoneme conversion process.

4.4.1 PRE-PROCESSING OF GRAPHEMIC NULLS

Many languages require few or no graphemic nulls and the additional variability introduced by catering for graphemic nulls result in miss-alignments. For our base algorithm (*Align v1*) we use forced Viterbi alignment based on the probabilities *P(grapheme i | phoneme j)*; and initialise probabilities from words and pronunciations that have equal length, as described by Andersen *et al* [54]. However, we insert graphemic and phonemic nulls in two separate steps. In a pre-processing phase, graphemic null generator pairs (two graphemes that result in more than two phonemes) are identified by Viterbi alignment of all word-pairs where pronunciation length is longer than word length. Phonemic nulls are inserted in a second phase of Viterbi alignment. (Where the first phase introduces unnecessary graphemic nulls, these are typically mapped to phonemic nulls during the second phase.) In both phases the alignment process is repeated until no further likelihood improvement is observed.

Alignment accuracy on the *NETtalk* corpus using this implementation (*Align v1*) is higher than the results reported by Andersen *et al* [22], as compared in Table 4.1. This improvement is due to an implementation difference rather than a conceptual difference: The algorithms are similar, apart from the different handling of graphemic nulls, and graphemic nulls do not occur in the *NETtalk* corpus[3].

4.4.2 UTILISING THE PHONEMIC CHARACTER OF NULL-PHONEMES

An additional improvement can be obtained if the transcription convention used by *NETtalk* is adapted. In *NETtalk*, null phonemes are used to identify graphemes that are "deleted" during pronunciation, for example the word *writer* is transcribed as $w\ r\ i\ t\ e\ r \rightarrow \phi\ r\ ay\ t\ \phi\ axr$. An alternative convention would be to use null phonemes simply to identify instances where two or more graphemes give rise to a single phoneme (without identifying a particular grapheme as deleted), by aligning the first grapheme in such a group with a non-null phoneme, and subsequent graphemes with nulls. Using this convention, the word *writer* is transcribed as $w\ r\ i\ t\ e\ r \rightarrow r\ \phi\ ay\ t\ axr\ \phi$. A null phoneme then simply indicates that the phonemic realisation remains the same for more than one grapheme.

Using a set of about 40 rewrite rules, the *NETtalk* dictionary can be rewritten using either the one convention or the other. Using the second convention, the dictionary responds better to data-

[3] In earlier work, when adding graphemic nulls by hand, we found that the use of pseudo-phonemes can complement the use of pseudo-graphemes. Pagel *et al* [52] suggested the use of pseudo-graphemes (e.g. creating two graphemes Xx to represent the k and s phonemes that originate from x separately). We found that, when a more natural choice, the use of pseudo-phonemes (e.g. creating a ks phoneme to represent the k and s combination) can improve alignment accuracy.

driven alignment and the second version of our Viterbi algorithm (*Align v2*). This algorithm explicitly calculates the *probability that a specific grapheme is realised as a null phoneme, given the previous non-null phonemic realisation of the preceding grapheme or graphemes*, and provides a significant performance improvement, as shown in Table 4.1.

Table 4.1: *Phoneme and word alignment accuracy obtained on the NETtalk corpus.*

Database	Type	Phoneme	Word
NETtalk-original	Iterative Viterbi [22]	93.2	83.7
NETtalk-original	Align v1	96.5	87.3
NETtalk-rewritten	Align v2	98.7	95.4

The effect of the improvement in alignment accuracy on rule extraction accuracy is depicted in Fig. 4.1. The *Align v1* and *Align v2* algorithms are used prior to *DEC-min*[4] rule extraction on a 10,000-word subset of the *FONILEX* database, and grapheme-to-phoneme prediction accuracy measured against a 5000-word test set.

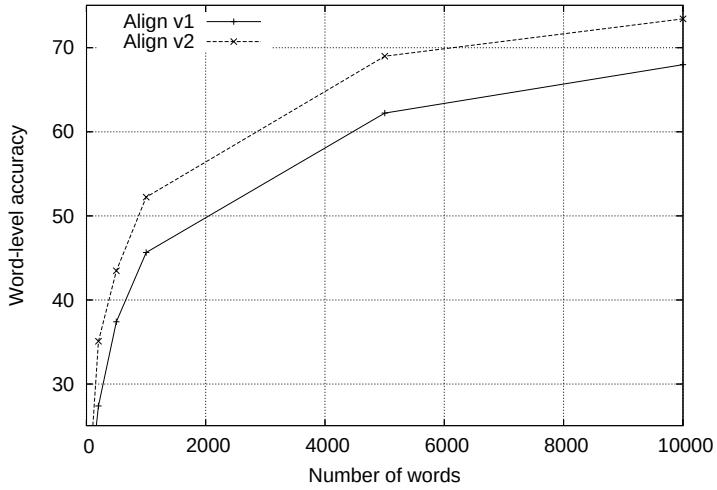

Figure 4.1: *Effect of different alignment algorithms on word-level pronunciation prediction accuracy of DEC-min, as measured on a 10,000-word subset of FONILEX.*

In order to verify that this effect is not corpus-specific, we perform a further evaluation using the *OALD* corpus. We analyse the effect of the two different alignment algorithms (*Align v1 and Align v2*) when extracting both *DEC-grow* and *DEC-min* rules using training sets of increasing size. For each training set of a specific size, 10 distinct training sets are generated. All training sets are tested against a non-overlapping 5970-word test set (10% of the full data set). A similiar trend is observed as on the *FONILEX* corpus, as depicted in Fig. 4.2. For example, the mean phoneme accuracy for *DEC-grow* rules trained on a 5000-word training set is 86.83% (with $\sigma_{10} = 0.07$) when aligned according to *Align v1*, and 87.54% (with $\sigma_{10} = 0.06$) when aligned according to *Align v2*. During the earlier

[4]The *DEC-min* algorithm is described in Section 4.5.2.

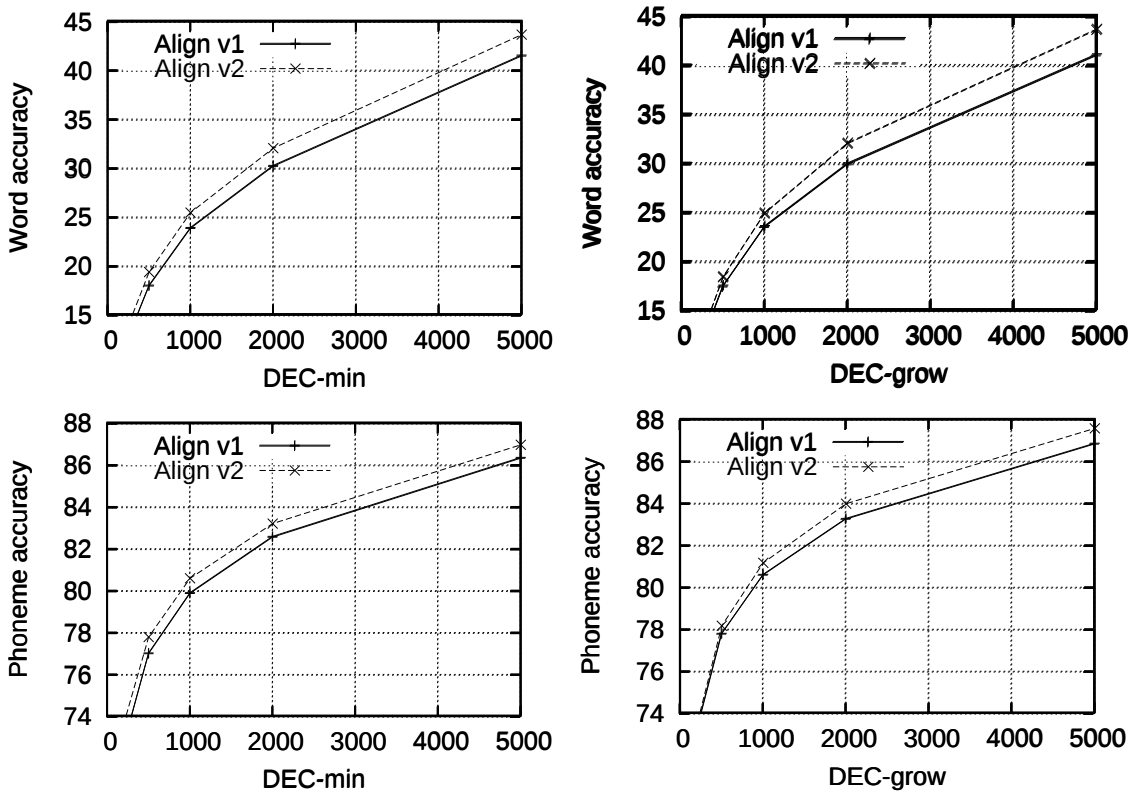

Figure 4.2: *Effect of different alignment algorithms on prediction accuracy of DEC-grow and DEC-min, as measured using the OALD corpus.*

stages of the rule extraction process (when alignment probabilities are still unstable) this provides a signficant advantage.

4.5 DEC-BASED GRAPHEME-TO-PHONEME PREDICTION

4.5.1 STANDARD DEC

A conceptual description of DEC as applied to the grapheme-to-phoneme problem by Torkkola [21] is provided in Section 2.2.2.3. In this section, we discuss the approach in further detail: Each DEC rule specifies a mapping of a single grapheme to a single phoneme for a given left and right graphemic context, i.e is of the form: *(left-context,grapheme,right-context)* $\rightarrow$ *phoneme*. Each word in the training dictionary is aligned with its pronunciation on a per-grapheme basis, as illustrated in Table 4.2. Rules are extracted by finding the smallest context that provides a unique mapping of grapheme to phoneme. If an $n-$letter context is not sufficient, the context is expanded to either the right or the left. This 'specificity order' influences the performance of the algorithm. Different orderings are illustrated in Table 4.3 as applied to grapheme *'s'* in the word *'interesting'*. Context 1 is expanded symmetrically on a right-grapheme-first basis, context 2 is expanded symmetrically on a left-grapheme-first basis, and context 3 favours the right context on a 2:1 basis. The set of extracted rules are stored as a hierarchical tree, with more general rules at the root, and more specific rules at the leaves. The tree is traversed from the root to the leaves, and the rule at the first matching leaf (the rule describing the

Table 4.2: *Word alignment and rule extraction in standard DEC.*

Alignment examples	r o s e → r ow z ϕ r o w s → r ow ϕ z r o o t → r uw ϕ t
Rule examples for -o-	in context -o: -o-o → uw in context -se: -o-se → ow in context o-: o-o- → ϕ

Table 4.3: *Different examples of context expansion order in DEC.*

size	context 1	context 2	context 3
0	s	s	s
1	st	es	st
2	est	est	sti
3	esti	rest	esti
4	resti	erest	estin

largest matching context) is used to predict the specific grapheme-to-phoneme realisation. If no leaf is matched, the most probable outcome of the last matching leaf is used, as can be estimated from the training data. In our implementation of DEC, we do not explicitly order the rules in a tree structure, but number them according to the order in which they are extracted (corresponding to a topological sort of all rules that can apply to a single word). We then search via reverse rule order rather than tree traversal. This variation does not change the algorithm functionally.

If DEC is not allowed to grow an asymmetric context when it reaches a word boundary and conflicting rules are ignored (*DEC-conflict*) the performance of the algorithm degrades for larger training corpora, especially if rules regarding the context surrounding a grapheme early or late in a word are of predictive importance. In order to remove this effect, the version of DEC (*DEC-grow*) that was implemented as baseline algorithm allows a context to grow towards the opposite side if a word boundary is encountered. This effect is illustrated in Fig. 4.3 where we plot the results for *DEC-conflict* and *DEC-grow* during the initial stages of learning (using the *FONILEX* corpus).

4.5.2 SHIFTING WINDOWS

DEC, as applied by Torkkola [21] expands the context of a grapheme one letter at a time, either favouring the right- or left-hand side explicitly. We analyse the implications of using a sliding window rather than a strict expanding context. We define a sliding window that first considers all possible contexts of size *n*, before continuing to consider contexts of size *n*+1, which prevents rules with unnecessarily large contexts from being extracted. In contrast to the DEC context expansion of Table 4.3, a sliding window applied to grapheme *'s'* in the word *'interesting'* would result in the context ordering indicated in Table 4.4. Since multiple rules of the same context size may apply to a single grapheme-to-phoneme mapping (such as *re,s,ti* → *s* and *ere,s,t* → *s*), contexts that are already served

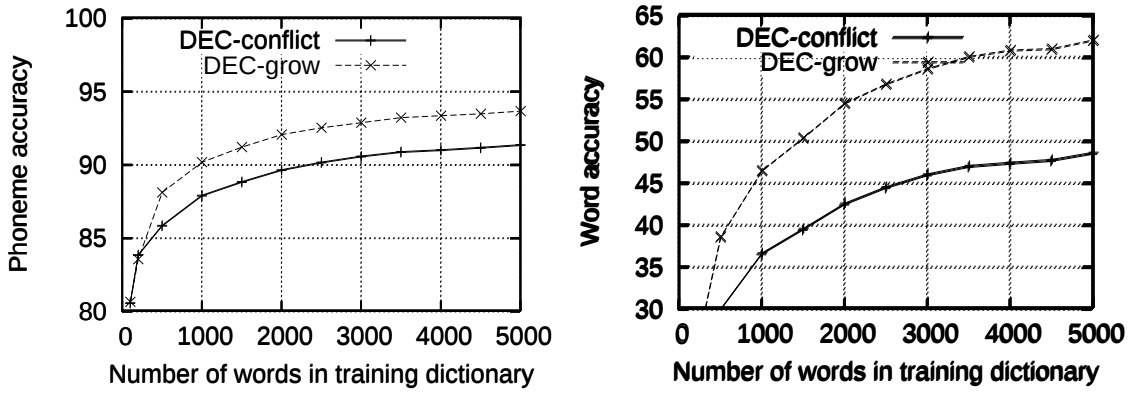

Figure 4.3: *Comparing DEC-conflict and DEC-grow during initial learning stage (first 5000 words of FONILEX). DEC-grow is chosen as baseline algorithm.*

Table 4.4: *Context expansion order in shifted DEC.*

order	size	context	order	size	context
1	0	s	2	1	st
3	1	es	4	2	est
5	2	sti	6	2	res
7	3	esti	8	3	rest
9	3	stin	10	3	eres

by existing rules can be removed to prevent over-specialisation. Because all contexts of each size are considered, the order in which contexts are expanded (for a specific context-level) becomes less significant than in standard DEC.

Figures 4.4 and 4.5 compare the performance of different DEC variations. In all experiments, a symmetric right-first expansion scheme is used[5] (as also in Table 4.4). The size of the maximum context allowed when extracting rules is not restricted, and the same word training order (random selection from corpus) is used. In order to compare with previous results, we use the exact alignments as used in [41]. Where word variants occur, we only use the first variant – both for training and testing purposes.

Three shifted window versions of DEC are implemented: extracting the first valid rule encountered (*DEC-win*) extracting the maximum number of valid rules (*DEC-max*) and pruning this system to obtain the minimum number of rules that still provide full coverage for the training corpus (*DEC-min*). When a shifting window is used, more than one conflicting rule of the same size may apply to a word. Various conflict resolution strategies can be implemented: in the set of experiments reported below, the most frequently observed rule is favoured. For the training set sizes analysed, the pruned, shifted window version of DEC (*DEC-min*) provides a small but consistent performance improvement in word accuracy[6]. *DEC-win* is not shown, but results in a learning curve similar to *DEC-grow*, both

[5] A symmetric, right-first expansion scheme is used when rule options are generated for consideration prior to selection of the actual rule – actual rules are generated according to a shifting window, and do not exhibit strict right-first behaviour.

[6] Note that phoneme accuracy initially follows a different trend for this corpus.

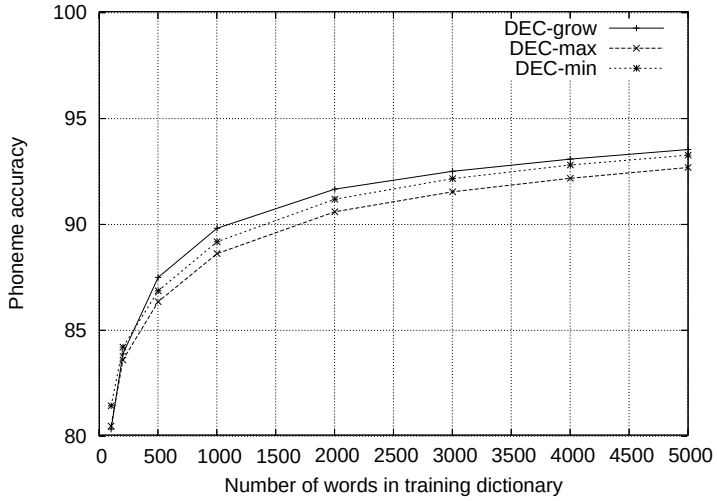

Figure 4.4: *Word-level accuracy of different DEC variations during initial learning stage, as measured using the first 5000 words of FONILEX.*

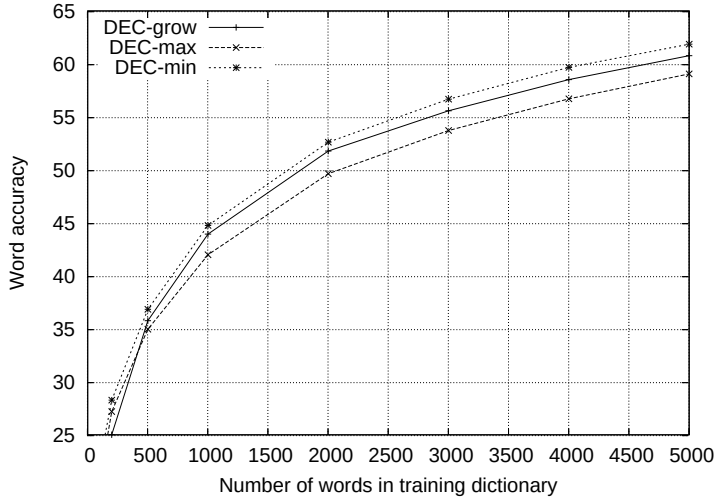

Figure 4.5: *Phoneme-level accuracy of different DEC variations during initial learning stage, as measured using the first 5000 words of FONILEX.*

with regard to word and phoneme accuracy. Asymptotic performance is only approached for larger training sets, as compared in Table 4.5. *DEC-min* continues to perform better than *DEC-grow*, with a small margin. The improvements during the initial learning stages are small, and introduce additional overhead during computation. Of more interest is that the new DEC variation (*DEC-min*) forms the basis for further algorithmic improvement, as discussed in the next sections.

As can be expected, the extracted rule sets grow in different ways with regard to rule number and rule length, as the size of the training dictionary increases. An analysis of the different types of rule sets extracted from the same training dictionary is provided in Table 4.6. The numbers of rules of each size (the size of the context that specifies the rule) are compared, as extracted from different sized training dictionaries using *DEC-grow*, *DEC-max* and *DEC-min*. Note that *DEC-max* tends to extract more rules than *DEC-grow* but that these rules tend to be shorter. *DEC-min* reduces

Table 4.5: *Phoneme correctness, phoneme accuracy and word accuracy comparison for different DEC variations, as measured using the FONILEX corpus.*

	phoneme correctness		phoneme accuracy		word accuracy	
		$\pm\sigma_{10}$		$\pm\sigma_{10}$		$\pm\sigma_{10}$
DEC-max	98.44	0.01	98.28	0.01	88.71	0.06
DEC-grow	98.50	0.01	98.32	0.04	88.60	0.07
DEC-win	98.57	0.01	98.40	0.01	89.53	0.05
DEC-min	98.58	0.01	98.41	0.01	89.58	0.06

the number of rules significantly (in comparison with *DEC-max*). *DEC-min* extracts slightly more rules than *DEC-win*, but as can be expected, these are much shorter (more general).

Table 4.6: *Number and size of rules: DEC-grow, DEC-max and DEC-min*

Rule type:	DEC-grow			DEC-max			DEC-min		
Dict size:	100	1,000	10,000	100	1,000	10,000	100	1,000	10,000
1	27	27	27	27	27	27	27	27	27
2	65	92	103	108	105	103	86	104	102
3	127	545	1259	256	1224	2375	102	705	1661
4	19	323	1996	24	926	7031	9	375	3469
5	7	131	1845	-	78	3081	-	33	1381
6	-	33	712	-	7	341	-	3	178
7	-	8	280	-	-	27	-	-	18
8	-	-	71	-	-	5	-	-	3
9	-	-	32	-	-	1	-	-	1
10	-	-	4	-	-	1	-	-	1
11	-	-	5	-	-	-	-	-	-
12	-	-	1	-	-	-	-	-	-
13	-	-	-	-	-	-	-	-	-
14	-	-	-	-	-	-	-	-	-
15	-	-	1	-	-	-	-	-	-
Total	245	1,160	6,337	415	2,367	12,992	224	1,247	6,841

4.5.3 RULE PAIRS

When analysing the specific errors made by these DEC variations, it becomes apparent that some rules occur in 'rule pairs', i.e. two rules always occur as companions in the training data. These rule pairs are sometimes not applied as companions in the test data, causing errors. For example, during rule extraction a rule $-e-en \rightarrow iy$ is typically followed by a second rule rule $e-e-n \rightarrow \phi$ or $e-e- \rightarrow \phi$, and is a better rule to apply when predicting the instance $k-e-en$ than the otherwise equally likely rule $\#k-e-$. We experiment with the implication of forcing such rule pairs to occur in tandem. First, we identify rule pairs that always occur together in the training data and exhibit a context overlap of at least the two focal graphemes. Then we restrict our rule application to only use one of the rules in such a pair if the second rule in the pair is also applicable to the

same training instance. However, constraining rule pairs in this way does not have a significant effect on predictive accuracy: In some instances the rule pair approach does correct a second phoneme that would otherwise have been wrong, but in a comparable number of cases this approach causes a second phoneme to be wrong, which would otherwise have been correct. We therefore do not continue with further experimentation along this route.

4.5.4 CONFLICT RESOLUTION

In standard DEC, the largest matching rule is always unique[7]. When a shifting window is used, more than one conflicting rule of the same size may apply to a word. If we use $num(r, p)$ to specify the number of training instances that match the context of a specific rule r and specific outcome as p, we calculate the 'accuracy' of the rule r as:

$$accuracy(r) = \frac{num(r, outcome(r))}{\sum_x num(r, x) + 1} \text{ for all possible outcomes } x. \tag{4.3}$$

In the experiments described above, if more than one candidate rule (of the same size) is applicable to the current word being predicted, we choose the rule for which $accuracy(r)$ is highest. This is a fairly simple conflict resolution strategy, and various alternative options are possible. We experiment with a number of these, including (1) voting among possible rules (choosing the outcome that most of the candidate rules agree upon), (2) applying the smaller (fall-back) context rather than any of the larger conflicting rules, and (3) simply choosing any of the rules at random (in practice whichever of the candidate rules was generated first during rule extraction), and find no consistent improvement using any of the alternative conflict resolution strategies. We continue to use the initial conflict resolution strategy (highest $accuracy(r)$) for further experimentation.

4.5.5 DEFAULT RULES

The question of how to best resolve conflict is closely linked to the question of how to best define default rules. One of the consequences of DEC rule extraction is that there exists only a single rule of any given length that can potentially apply to a specific word (where this length lies between one and the total length of the word being predicted). If the word being predicted is of length n, and no matching rule of length n exists, then a single rule of size $n - 1$ may potentially apply. In effect the latter rule acts as 'back-off value' for the rule of length n. If a rule of length $n - 1$ does not exist either, the (unique) matching rule of length $n - 2$ becomes the next possible candidate[8]. When using shifting windows, there is no longer a unique rule of any given length that can potentially apply when predicting a word – more than one candidate may exist. We therefore consider the effect of adding default rules explicitly: for any set of rules of context size n with one or more internal disagreements

[7] This rule may be conflicted (i.e. not a leaf node in Torkkola's original implementation) in which case the most frequently observed outcome across the training data is generated, but no conflicting rules of the same size can exist.

[8] From a conceptual perspective – this is not the process that is followed in practice during DEC prediction.

and no 'default rule' of size $n-1$, we add an explicit rule of context size $n-1$ with an outcome p such that $num(r,p)$ is the maximum over all possible outcomes. Interestingly, adding this additional information decreases rule accuracy. An error analysis indicates that inappropriate 'default rules' are extracted: while these rules correctly 'fill the gaps' among the rules extracted from the training data, the 'default rules' are forced to specific value by the previously extracted *DEC-min* rules, and do not generalise well. This leads us to the definition of a default-and-refinement approach to grapheme-to-phoneme prediction, as discussed in the next section (Section 4.6). This approach utilises a similar rule definition format as DEC, but the rule extraction process is more distant from original DEC than the variations studied up to this point.

4.6 A DEFAULT-AND-REFINEMENT APPROACH TO G-TO-P PREDICTION

Grapheme-to-phoneme prediction algorithms rely on the connection between the spoken and written form of a language. It is expected that, the more modern the writing system of a language, the stronger this connection, and the more regular the spelling system of the language [71]. This may not always hold in practice, for example, when a language with mainly (or only) an oral tradition is transcribed for the first time, and the variability introduced through the initial transcription process has not yet stabilised through usage or an education system that utilises the written form. While alternative outcomes are possible, the languages studied as part of this thesis all exhibit a combination of a fairly modern writing system associated with a fairly to highly regular spelling system.

The more regular the spelling system of the language, the stronger the concept of a 'default phoneme': a grapheme that is realised as a single phoneme significantly more often than as any other phoneme. Figure 4.6 and Figure 4.7 illustrate this phenomenon for Flemish. When counting the number of times a specific grapheme is realised as a specific phoneme, most graphemes follow the trend depicted in Figure 4.6. Here, *y* is realised as a single phoneme more than 60% of the time, with the next two phonemic candidates occurring only 24% and 4% of the time, respectively. For graphemes that exhibit 'conflicted default phoneme' behaviour, such as (*h*,*j*,*n*,*u*), the trend is less strong, but also clearly discernible, as depicted in Figure 4.7. Similar trends are observable for languages with less regular spelling systems, with a larger proportion of graphemes of these languages displaying the behaviour depicted in Figure 4.7.

We use this information to define an algorithm that utilises greedy search to find the most general rule at any given stage of the rule extraction process, and explicitly orders these rules according to the reverse rule extraction order[9]. Explicitly ordering the rules provides flexibility during rule extraction, and ensures that the default pattern acts as a back-off mechanism for the more specialized rules. The framework we use is similar to that used in previous sections: Each grapheme-to-phoneme rule

[9]It is interesting to note that, while the rule application order of DEC is ordered by context size (largest rule first), our reverse rule extraction order automatically reverts to context size ordering in the case of DEC-based rule extraction.

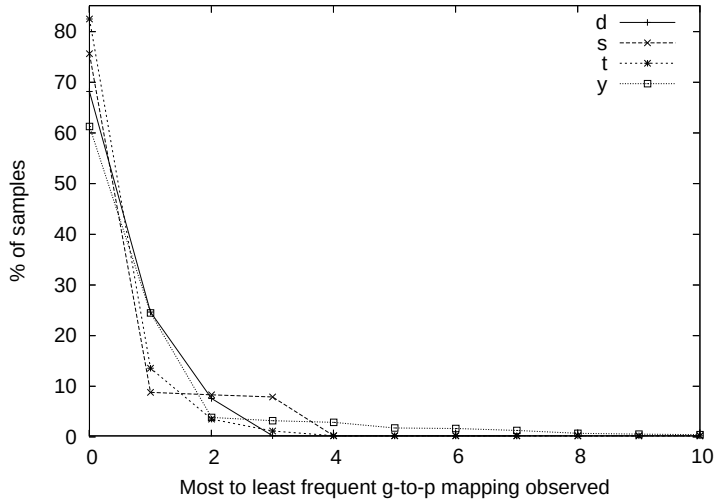

Figure 4.6: *Default phoneme behaviour of graphemes d,s,t and j in Flemish. Only the first 10 phonemic candidates are displayed.*

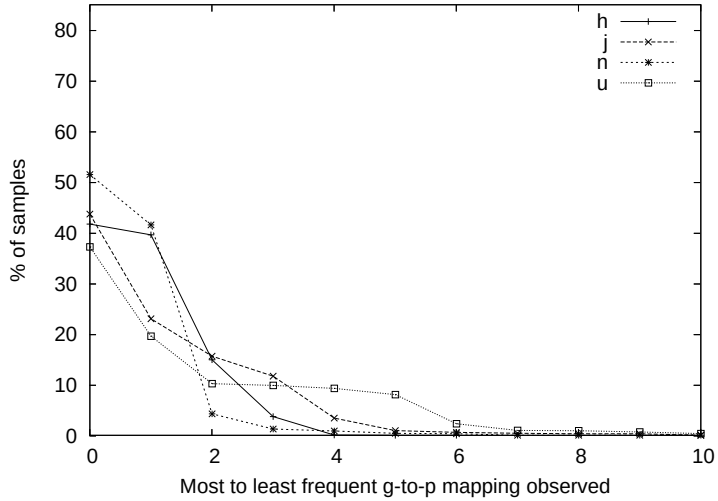

Figure 4.7: *Conflict phoneme behaviour of graphemes h,j,n,u in Flemish. Only the first 10 phonemic candidates are displayed.*

consists of a pattern

$$g_{left} - g - g_{right} \rightarrow p \tag{4.4}$$

where g indicates the grapheme being considered, g_{left} and g_{right} are the graphemic left and right contexts of the rule, and p the specific phonemic realisation of g. The pronunciation for a word is generated one grapheme at a time. Each grapheme and its left and right context as found in the target word are compared with each rule in the ordered rule set; and the first matching rule is applied.

Prior to rule extraction, grapheme-to-phoneme alignment is performed according to the Viterbi-based alignment process described in Section 4.4. Pronunciation variants are currently not allowed: if a word has more than one possible pronunciation, only the first is kept. Each aligned word-pronunciation pair is used to generate a set of possible rules by extracting the sub-pattern of each word pattern; an example of such a process is shown in Table 4.7.

Table 4.7: *The relationship between a word (test) and, for one of its graphemes (e), the word pattern and sub-patterns that are generated during rule extraction.*

Word	test
Word pattern	#t-e-st# → eh
Sub-patterns	-e- → eh,-e-s → eh,t-e- → eh,t-e-s → eh t-e-st → eh, #t-e-s → eh,-e-st# → eh #t-e-st → eh,t-e-st# → eh,#t-e-st# → eh

Once all possible rules have been generated in this way, rules are extracted on a per-grapheme basis, one rule at a time. For any specific grapheme, applicable words are split into two sets based on whether the current rule set (initially empty) predicts the pronunciation of that grapheme accurately (*Completed* words) or not (*New* words). These two large word sets are used to keep track of status, but further manipulation utilises two sets of sub-patterns: the *Possible* sub-patterns, indicating all possible new rules, and consisting of all the sub-patterns of each word pattern in *New*, excluding all for which the left-hand side is an existing rule; and the *Caught* set of sub-patterns, indicating all the sub-patterns covered by the current rule set irrespective of whether the outcome of the rule matches that of the word or not. Both the *Possible* and *Caught* sets of sub-patterns count the number of times, per possible outcome, that a matching word pattern is observed in the relevant word sets.

The next rule is chosen by finding the pattern for which the matching count in *Possible* minus the conflicting count in *Caught* is highest. (The conflicting count is the number of times a matching left-hand pattern is observed with a conflicting right-hand phoneme.) Definition of a new rule moves words from the *New* to the *Completed* set. Any words that are currently in the *Completed* set and conflict with the new rule, are moved back to the *New* set. This process is repeated until all words have been moved from the *New* to the *Completed* set. The algorithm ensures that the next rule chosen is the one that will cause the most net words to be moved from the *New* to the *Completed* set, irrespective of context size. As this number (net words processed) is always positive[10], the algorithm cannot enter an infinite loop. The stronger the default behaviour exhibited by a specific grapheme described by a new rule, the more words are processed during the extraction of that specific rule. Conflict is only resolved in the *Completed* set: new rules are allowed to conflict with words still in *New*, which ensures that the rule set is built for the default pattern(s) first.

In order to ensure computational efficiency when trained on larger dictionaries, we use the following techniques during implementation:

- Words are pre-processed and the word patterns relevant to a single grapheme extracted and written to file. All further manipulation considers a single grapheme (and the corresponding set of word patterns) at a time.

- The context size of the sub-patterns considered is grown systematically: only sub-patterns up to

[10] A rule based on a full word pattern can only apply to that single word, and will result in a 'net move count' of 1. Since the maximum of all these 'net move counts' is selected, this value will always be positive.

size $max + win$ are evaluated, where max indicates the current largest rule, and win is defined to ensure that any larger contexts that may be applicable are considered, without requiring all patterns to be searched.

- Whenever a sub-pattern in *Possible* or *Caught* reaches a count of zero, the sub-pattern is deleted and not considered further, unless re-added based on an inter-set move of a related word.

While these techniques ensure that a fairly large dictionary (200,000 words) can be trained in an acceptable amount of time when using the process in a non-interactive fashion, the process to train a sizeable dictionary becomes too slow for interactive bootstrapping. This issue is addressed further in Section 4.6.4. In the remainder of this thesis we refer to the algorithm described above as '*Default&Refine*'.

4.6.1 ASYMPTOTIC PERFORMANCE

In order to evaluate the asymptotic behaviour of *Default&Refine*, we compare our results on a fairly large corpus with published results for a number of alternative algorithms. As the *Default&Refine* algorithm is motivated by 'default behaviour', we first evaluate the algorithm on a language with a fairly regular spelling system (Flemish), before testing it on a language with an irregular spelling system (English).

4.6.1.1 REGULAR SPELLING SYSTEMS

We evaluate the accuracy of the *Default&Refine* algorithm when trained on the full *FONILEX* training set, and compare its performance with that of alternative algorithms in Table 4.8: the *IB1-IG* result utilises an instance-based learning algorithm and is as reported in [41]; the *DEC-grow* and *DEC-min* results are calculated using the algorithms described in Section 4.5.2; and the *D&R* result reports the *Default&Refine* values. The *DEC* and *Default&Refine* experiments utilise the same alignments as used in [41].

Table 4.8: *Phoneme correctness, phoneme accuracy and word accuracy comparison for different algorithms using the FONILEX corpus*

	phon correct		phon accuracy		word accuracy	
		$\pm\sigma_{10}$		$\pm\sigma_{10}$		$\pm\sigma_{10}$
IB1-IG	98.18	-	-	-	86.37	-
DEC-grow	98.50	0.01	98.32	0.04	88.60	0.07
DEC-min	98.58	0.01	98.41	0.01	89.58	0.06
D&R	98.87	0.01	98.78	0.01	92.03	0.06

The focus of [41] was to investigate the effect of cascading two classifiers – one trained on *FONILEX* and one on *CELEX* – a Dutch variant corpus, and creating meta-classifiers using C5.0 (decision tree learning), IB1-IG (instance-based learning as described in Section 2.2.2.3), IGTREE (an algorithm

that induces decision trees utilising information gain) and MACCENT (a maximum entropy-based algorithm). The highest accuracy reported was for such a meta-classifier system: 91.55% word accuracy for a single meta-classifier; and 92.25% word accuracy for a meta-meta-classifier of all meta-classifiers. (These systems all utilised the $CELEX$ data as an additional data source.) We find that *Default&Refine* has good asymptotic accuracy, and performs better than the comparative (single) classifiers.

4.6.1.2 LESS REGULAR SPELLING SYSTEMS

As the algorithm is motivated by 'default behaviour' we were interested in the extent in which the algorithm would fail for a language such as English, with a less regular spelling system. We therefore evaluate the asymptotic performance of the algorithm against benchmark results available for both the *NETtalk* and the *OALD* corpus. It is reassuring to find that the algorithm again performs well, as shown in Tables 4.9 and 4.10.

Table 4.9: *Phoneme accuracy, phoneme correctness and word accuracy comparison for different algorithms using the NETtalk corpus*

	phon correct		phon accuracy		word accuracy	
		$\pm\sigma_{10}$		$\pm\sigma_{10}$		$\pm\sigma_{10}$
Trie	-	-	89.8	-	51.7	-
DTree	-	-	89.9	-	53.0	-
DEC-T	-	-	90.8	-	-	-
DEC-Y	-	-	92.21	-	56.67	-
D&R	91.37	0.08	90.50	0.1	58.66	0.21
SMPA	-	-	93.19	-	63.96	-

In Table 4.9 we compare the performance of a number of algorithms on the *NETtalk* corpus. We list the results obtained by Andersen *et al* [22] using Trie structures (*Trie*) and decision trees (*DTree*) respectively; by both Torkkola [21] and Yvon [36] using Dynamically Expanding Context (*DEC-T* and *DEC-Y*); by Yvon [36] using SMPA, a pronunciation-by-analogy algorithm; and the results of *Default&Refine* (*D&R*) using own alignments. The phoneme correctness reported in [36] for DEC seems anomalously high, in relation to our own experiments, those obtained in [21], and the reported word accuracy. The *SMPA* algorithm employs a pronunciation by analogy approach, and is less suitable for training on very small data sets. The latter results only pertain to words that could be pronounced – about 0.5% of words were not pronounceable with SMPA when fully trained. Note also that the SMPA results score the accuracy of variants in the test set differently to the approach employed in this thesis[11].

In Table 4.10 we compare the performance of *Default&Refine* (*D&R*) with the results obtained by

[11] In the SMPA experiments all variants but one are removed from the training set, but all variants are retained in the test set – if any of the possible variants are generated during testing, the prediction is marked as accurate. This is different to the scoring approach used in this thesis, as described in Section 4.3

Black *et al* [23] using Classification and Regression Trees (*CART*) for two data sets: one including stress assignment (SA) and one without. We use the exact alignments, and the same single training set and test set as used by Black[12]. The CART trees were generated taking part-of-speech information into account – which *Default&Refine* does not use. Without POS information, the CART result (with stress assignment) decreases to 95.32% phoneme correctness and 71.28% word accuracy .

Table 4.10: *Phoneme accuracy, phoneme correctness and word accuracy comparison for CART and Default&Refine using the OALD corpus (SA indicates stress alignment)*

	phon correct	phon accuracy	word accuracy
Incl. SA:			
CART	95.80	-	74.56
D&R	97.12	96.87	83.76
Excl. SA:			
CART	96.36	-	76.92
D&R	97.80	97.56	87.40

4.6.2 LEARNING EFFICIENCY

In order to use this algorithm for the bootstrapping of pronunciation dictionaries, we are specifically interested in the performance of the algorithm when trained on very small training sets. We therefore evaluate word and phoneme accuracy for different training dictionaries of sizes smaller than 3,000 words, using subsets from *FONILEX*. Figure 4.8 demonstrates the phoneme accuracy learning curve for *Default&Refine* in comparison with *DEC-grow*. Each rule set is evaluated against the full 17,387-word test set.

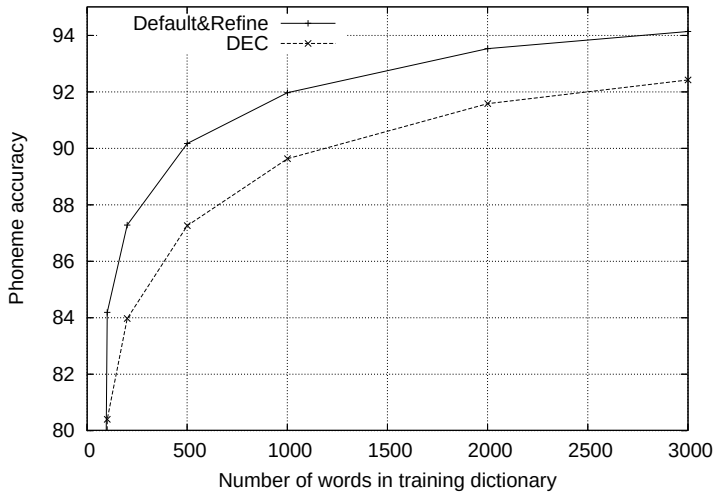

Figure 4.8: *Phoneme accuracy during initial 3000 training words, as measured using the FONILEX corpus.*

[12]When 10-fold cross-validation is performed using different subsets of this data set, a slightly lower cross-validated accuracy is obtained: 96.62% phoneme accuracy and 82.37% word accuracy when stress assignment is included, and 97.66% phoneme accuracy and 86.41% word accuracy without stress assignment.

4.6.3 SIZE OF THE RULE SET

While the size of the rule set is typically not a concern during grapheme-to-phoneme bootstrapping, it can be important for other applications (such as dictionary compression). We therefore analyse the size of the rule set, and find that the rule set extracted by *Default&Refine* is significantly smaller that extracted by *DEC-grow*, as shown in Figures 4.9 and 4.10. *Default&Refine* provides both a more accurate and more compact prediction model: the 156,486-word training dictionary is represented with 100% accuracy by 15,053 rules.

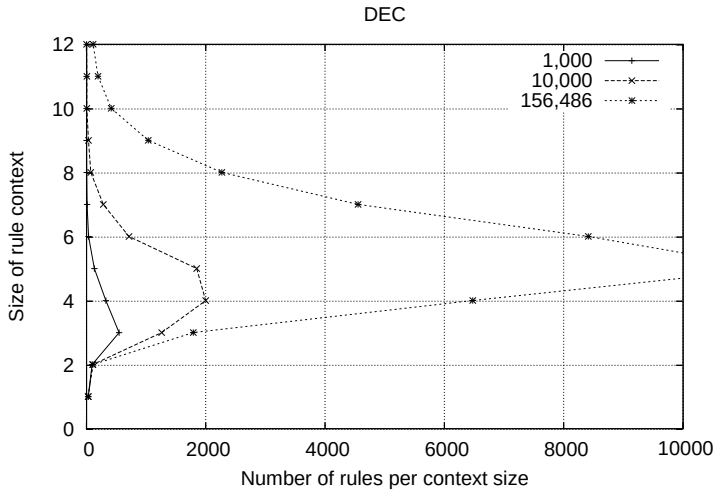

Figure 4.9: *Number of rules per context size extracted by DEC-grow from training dictionaries of three different sizes.*

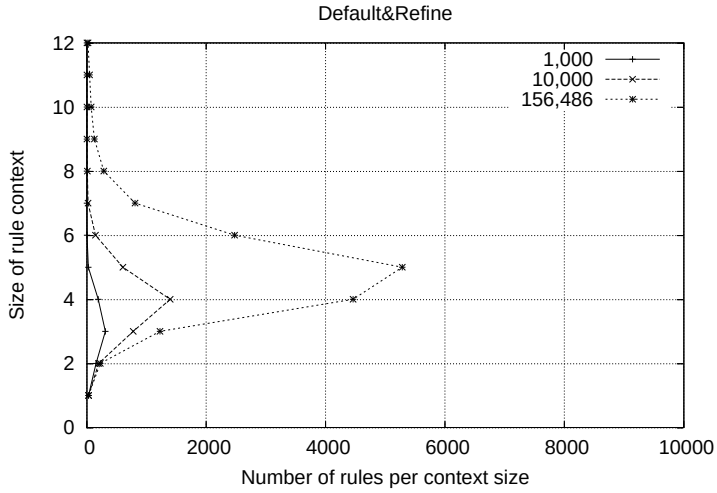

Figure 4.10: *Number of rules per context size extracted by Default&Refine from training dictionaries of three different sizes.*

4.6.4 CONTINUOUS LEARNING

The ideal bootstrapping system will be able to update the rule set after every correction by the verifier, immediately incorporating further learning in the bootstrapping knowledge base. The time taken for

such updates is therefore of crucial importance. The update speed is influenced by two factors: the alignment speed and the rule extraction speed. If n represents the number of words in the training dictionary, then the complexity of the alignment process and that of the rule extraction process is both approximately $O(n)$, if it is assumed for the sake of simplicity that all words are more or less of equal length[13]. This is typical of various of the rule extraction techniques that are appropriate for grapheme-to-phoneme bootstrapping.

If the entire set of training words is processed after every correction, the update time becomes a limiting factor as the dictionary grows. In our implementation, continuous updating becomes unwieldy when the number of words with known pronunciations exceeds approximately 2000. On the other hand, by performing batch updates at specific times that suit the verifier (e.g. at the end of a verification session), the update time does not become a constraint, but the learning obtained during the session is not utilised to refine models until after the end of the session. In order to obtain an algorithm that allows for continuous model updating while keeping the update time within acceptable limits, an incremental version of the *Default&Refine* algorithm was developed.

While the original algorithm creates a set of graphemic rule trees (one tree per grapheme) from the training set by considering all the training words simultaneously, the incremental version utilises the trees constructed during the previous (batch mode) update, and adds the new refinements as leaves to these trees: for each grapheme in the new word, if the realised phoneme is predicted accurately by the current graphemic tree, no update occurs; otherwise the smallest rule is extracted that will describe the new word without affecting any of the existing predictions. This version has $O(d)$ complexity where d represents the average depth of the various graphemic rule trees (which is approximately equivalent to the average context size of the graphemic rule set). Using this incremental process, additional learning can be obtained from the new words added without causing discernible delay, even for large training dictionaries.

In practice, the bootstrapping process operates in two phases: during the first phase a batch update occurs for every word; during the second phase a batch update occurs at synchronisation events only, and incremental updates are performed in between synchronisation events. The interval between synchronisation events is based on a set number of "update words", i.e. words that have been corrected by the verifier (words that were correctly predicted prior to verification do not contribute to this count). At the end of this interval, a synchronisation event occurs: the complete training dictionary is re-aligned, and new rules are extracted in batch mode. During the update interval, the Viterbi probabilities calculated at the previous synchronisation event are used per word to perform a fast alignment (the probabilities are used in the standard way, but not updated) and incremental *Default&Refine* is used to extract additional rules from the single aligned word-pronunciation pair. Phase 2 is initiated well before the time required by the full update event becomes noticeable. (For our current system we progress from phase 1 to phase 2 when 1500 valid words have been processed.)

As can be expected, the new algorithm is an approximation of standard *Default&Refine*, and

[13] See section 4.7.3 for a further discussion of the computational complexity of *Default&Refine.*

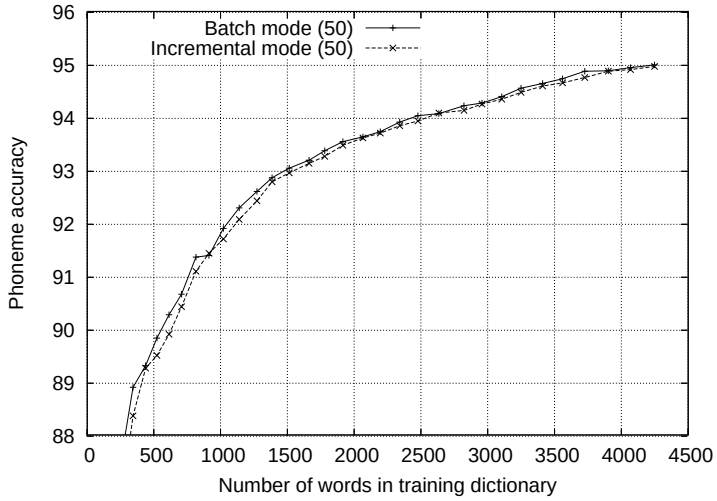

Figure 4.11: *Phoneme accuracy comparison for incremental and batch mode at an update interval of 50, measured using the FONILEX corpus.*

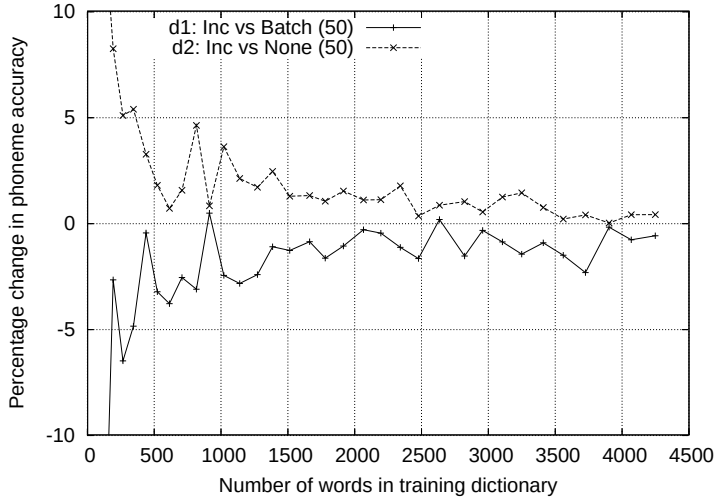

Figure 4.12: *Relative change in phoneme accuracy when comparing incremental with batch mode (δ_1), and incremental mode vs no updating between synchronisation events (δ_2); both at update interval 50.*

therefore somewhat less accurate than the original. We evaluate the performance of the system using an existing pronunciation dictionary (*FONILEX*), and perform 10-fold cross-validation on all our results. In order to determine the efficiency of the incremental approach, we first compare the two rule extraction processes (incremental mode and batch or standard mode) without taking changes in alignment into account. We utilise the same set of alignments[14] for both types of rule extraction, and measure phoneme accuracy on the same training set using the two different algorithms. We find that the decrease in accuracy is slight once the graphemic trees are of sufficient size, as demonstrated in Fig. 4.11 for a synchronisation interval of 50. The difference in accuracy can be analysed in further detail by calculating two values: δ_1, the relative increase in phoneme error rate when utilising the

[14]The alignments used were obtained from a 173,873-word training dictionary.

incremental mode compared to the batch mode, and δ_2, the relative decrease in phoneme error rate when utilising the incremental mode, in comparison with only performing updates at synchronisation events and not updating the models in between; that is,

$$\delta_1(x) = \frac{inc(x) - batch(x)}{1 - batch(x)} * 100 \tag{4.5}$$

$$\delta_2(x) = \frac{inc(x) - batch(x-1)}{1 - batch(x-1)} * 100 \tag{4.6}$$

and where $batch(x)$ indicates the phoneme accuracy using batch rule extraction, and $inc(x)$ the phoneme accuracy using incremental rule extraction, both at synchronisation point x. Fig. 4.12 illustrates the trends for the δ_1 and δ_2 values for an update interval of 50 (still utilising ideal alignments), providing an additional perspective on the same data as displayed in Fig. 4.11.

The effect on rule set accuracy is strongly influenced by the length of the update interval. We therefore compare the performance of the two algorithms for different update intervals, and find that the average δ_1 and δ_2 values are both fairly linear in relation to the update interval: the longer the interval, the less accurate incremental updating becomes when compared with batch updating, and the more value is provided by incremental updating vs performing no updates in between synchronisation events. In Fig 4.13 we plot the δ_1 and δ_2 values for update intervals of length $50, 100, 150$ and 200 during the first 4500 words of bootstrapping. These trends continue for larger update intervals.

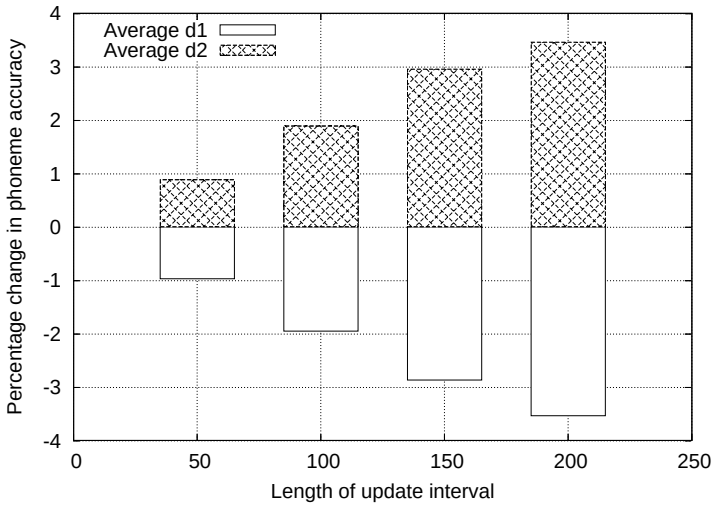

Figure 4.13: *Average δ_1 and δ_2 values for update intervals of length 50,100,150 and 200.*

Finally, in order to ensure that the fast alignment process does not introduce a noticeable loss in accuracy, we compare the two algorithms (batch and incremental rule extraction), applying the alignment process as it would be used in practise: performing a full alignment during synchronisation events and using the fast alignment process in between. We find that while there is a greater variance in the effect on phoneme accuracy when using the fast alignment process during the first phase of bootstrapping, this effect becomes negligible during the second phase of bootstrapping. (In practice, fast alignment is only used during the second stage of bootstrapping.) In Fig 4.14 we plot the δ_1

values for an update interval of 50, when using ideal alignments and actual alignments.

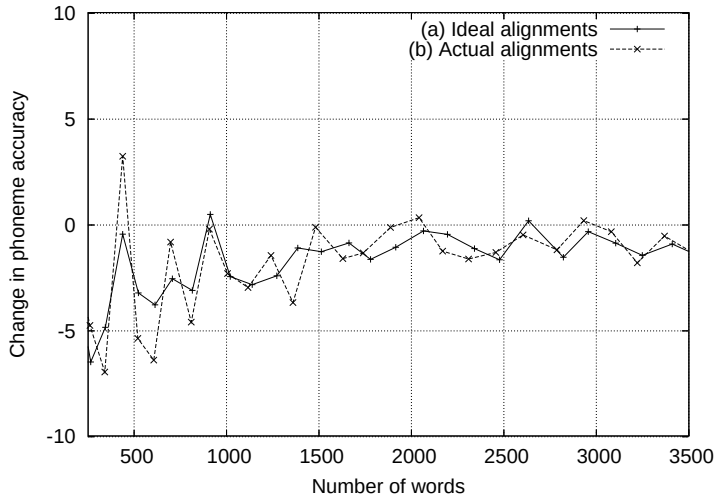

Figure 4.14: *Change in phoneme accuracy (δ_1) when comparing incremental with batch mode when (a) ideal alignments are used, and (b) when actual alignments are used.*

The above results indicate that incremental *Default&Refine* provides an effective way of increasing system responsiveness. As there is a clear trade-off between the length of an update interval and learning efficiency, the update interval can be chosen in a way that is suitable for the specific dictionary developer: longer continuous sessions (requiring slightly more corrections), or shorter sessions with frequent breaks. As the dictionary size increases and the rule set approaches asymptotic accuracy, the number of words considered between synchronisation events increases automatically[15]. For large dictionaries, the batch update process can become a daily event, rather than an hourly event, as would be the case for relatively small dictionaries. Also, as the user interface requires little processing capacity, the batch update may be scheduled to occur in the background during incremental verification, transparent to the user[16].

4.7 BOOTSTRAPPING ANALYSIS

In this section we summarise the characteristics of *DEC-grow*, *DEC-min* and *Default&Refine* according to the four main requirements for bootstrapping, as described in Section 4.2: predictive ability, conversion accuracy, computational cost and robustness to noise.

4.7.1 PREDICTIVE ABILITY

In Fig. 4.15 we compare the accuracy of the three algorithms for small training sets, using the *FONILEX* corpus. The *Default&Refine* algorithm performs particularly well, achieving 90%

[15]For example, using an update interval of 50, approximately 200 training words are considered per session when just past the 4000-word mark. (See Fig. 4.12.)
[16]This approach was not implemented.

phoneme accuracy prior to the 500 word-mark. *DEC-grow* requires an additional 800 words before the same level of accuracy is reached. Since the correction of incorrectly predicted phonemes is the most labour-intensive aspect of bootstrapping pronunciation dictionaries (as discussed in Section 6.3.2.4) this represents a significant improvement to the process.

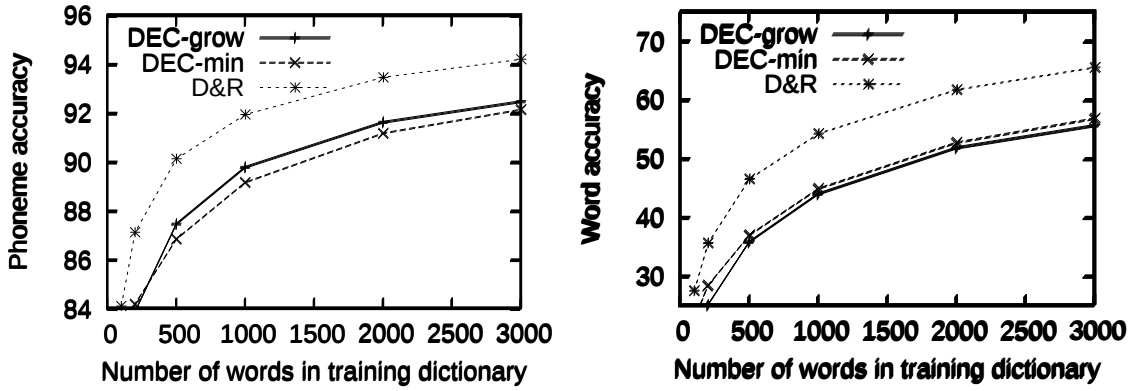

Figure 4.15: *Accuracy comparison during initial 5000 words of training, as measured using the FONILEX corpus.*

From a bootstrapping perspective, asymptotic accuracy is not as important, unless very large dictionaries are built. Asymptotic accuracies for different languages are compared for the dictionaries listed in Table 4.11. Per dictionary the number of words in total (*size*) and number of distinct words (*distinct*) are indicated. Word accuracy is listed in Table 4.12 and phoneme accuracy in Table 4.13, as analysed during 10-fold cross-validation of the dictionaries.

Table 4.11: *Dictionaries used for accuracy analysis*

Language	Dictionary	Size	Distinct
Afrikaans	Afrikaans B	7,782	7,782
English	NETtalk	20,008	19,802
English	OALD (no SA)	60,399	59,835
Flemish	FONILEX	173,873	163,526

Table 4.12: *Word accuracy of g-to-p algorithms for larger dictionaries in different languages.*

Dictionary	DEC-grow		DEC-min		Default&Refine	
		$\pm\sigma_{10}$		$\pm\sigma_{10}$		$\pm\sigma_{10}$
Afrikaans B	79.08	0.44	79.90	0.51	84.82	0.29
NETtalk	47.82	0.41	47.61	0.35	58.66	0.21
OALD (excl SA)	77.62	0.17	79.98	0.17	86.41	0.15
FONILEX	88.60	0.07	89.58	0.06	92.03	0.06

4.7.2 CONVERSION ACCURACY

All the studied algorithms are memory-based and provide complete retrieval of training data: the entire training dictionary can be reconstructed from the grapheme-to-phoneme rule set without any

Table 4.13: *Phoneme accuracy of g-to-p algorithms for larger dictionaries in different languages.*

Dictionary	DEC-grow		DEC-min		Default&Refine	
		$\pm\sigma_{10}$		$\pm\sigma_{10}$		$\pm\sigma_{10}$
Afrikaans B	95.96	0.09	95.98	0.14	97.08	0.08
NETtalk	87.82	0.11	87.20	0.08	90.50	0.10
OALD (no SA)	95.85	0.04	96.08	0.04	97.41	0.03
FONILEX	98.32	0.04	98.41	0.01	98.78	0.01

loss of accuracy.

4.7.3 COMPUTATIONAL COST

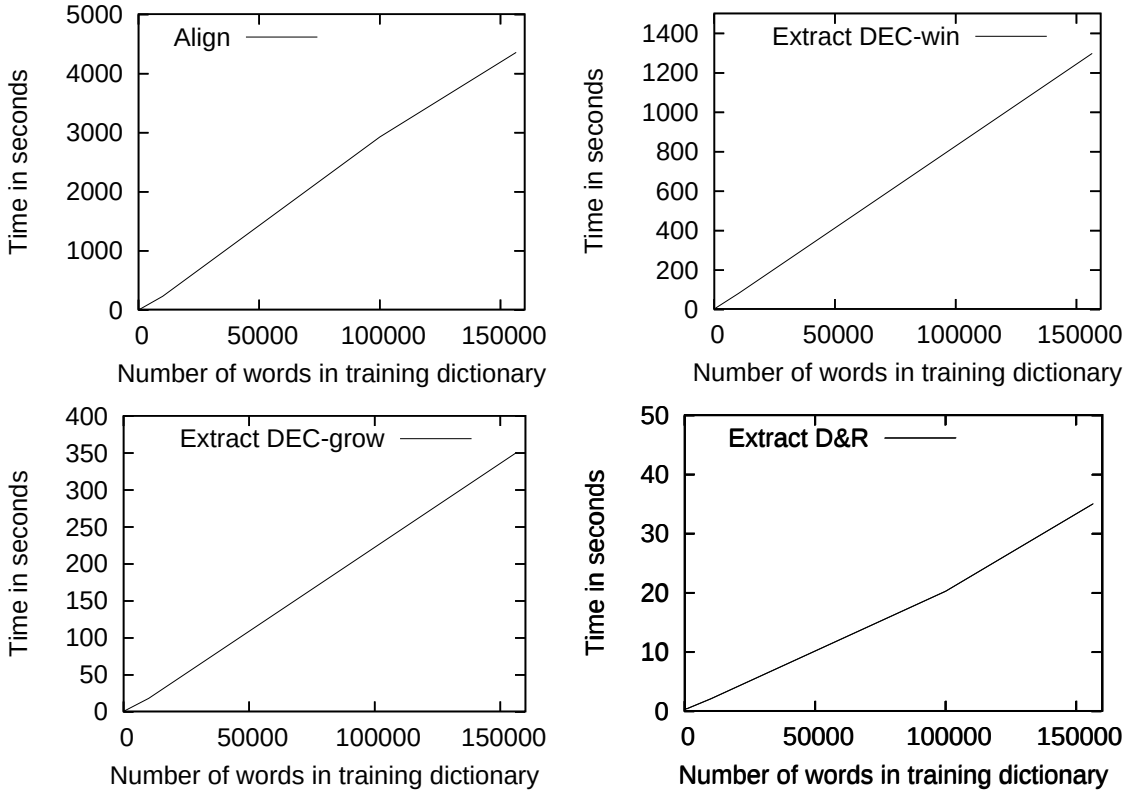

Figure 4.16: *Time required for alignment and extraction of initial patterns from different sized training dictionaries, measured using the FONILEX corpus.*

The computational cost of the various algorithms results from four separate processes:

1. *Grapheme-to-phoneme alignment:* Aligning words on a grapheme-to-phoneme basis. An identical grapheme-to-phoneme alignment process is used for all of the algorithms. The computational cost of alignment is influenced by the number of times the full dictionary is processed before the alignment probabilities stabilise. As this is typically a small number, alignment is approximately $O(n)$ where n indicates the number of words in the training dictionary. As the probabilities stabilise more quickly when more training data is available, alignment can exhibit better than linear dependency in practice.

2. *Extracting initial patterns prior to rule set extraction:* In the current implementation, the dictionary is read once, all required patterns are extracted and separated according to grapheme. Further rule extraction utilises the per-grapheme pattern sets as input. This process is again $O(n)$ for all the algorithms.

3. *Rule extraction:* Extracting a specific rule set from a grapheme-specific pattern set. For the implementations of *DEC-grow* and *Default&Refine*, rule extraction may require as many as

$$n + (n-1) + (n-2)... \sim \frac{n(n-1)}{2} \tag{4.7}$$

steps, which results in $O(n^2)$ behaviour. This would be the case for a dictionary that is conflicted to the extent that every single word gives rise to a separate rule. However, in practise, the number of steps required is closer to

$$n + k.n + k^2.n + ... \sim \frac{n}{1-k} \tag{4.8}$$

where $0 \leq k < 1$ provides some indication of the pronunciation conflict for the specific language (and dictionary) being considered. The more exceptions in the dictionary, the higher k, and the higher the complexity of rule extraction. In practice, rule extraction therefore displays $O(n)$ behaviour.

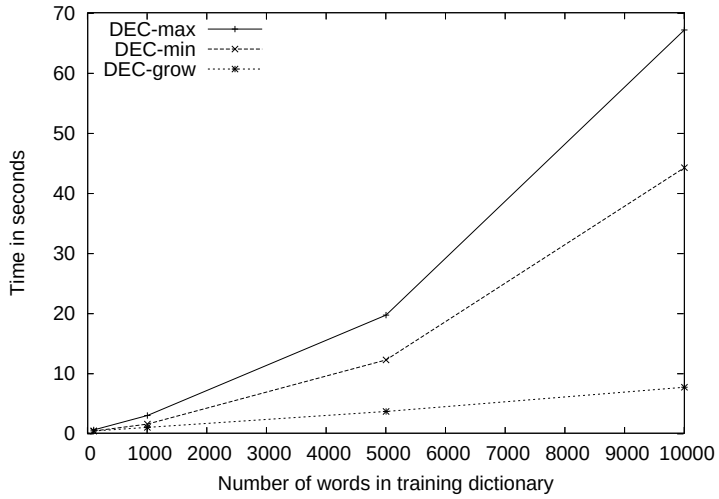

Figure 4.17: *Time required to extract DEC-grow, DEC-max and DEC-min rules from different sized training dictionaries, measured using the FONILEX corpus.*

4. *Pronunciation prediction:* Predicting the pronunciation of a single word based on an existing rule set. Pronunciation prediction is efficient for all the algorithms studied. For each type of rule extraction, the ensuing rule set can be arranged in an efficient tree structure. Pronunciation prediction is of $O(d.l)$ where l indicates the length of the word predicted, and d again represents the average depth of the various graphemic rule trees (which is approximately equivalent

to the average context size of the graphemic rule set, as described in Section 4.6.4). Our implementation of *DEC-max* and *DEC-min* exhibit worse than linear dependency, as depicted in Fig. 4.17.

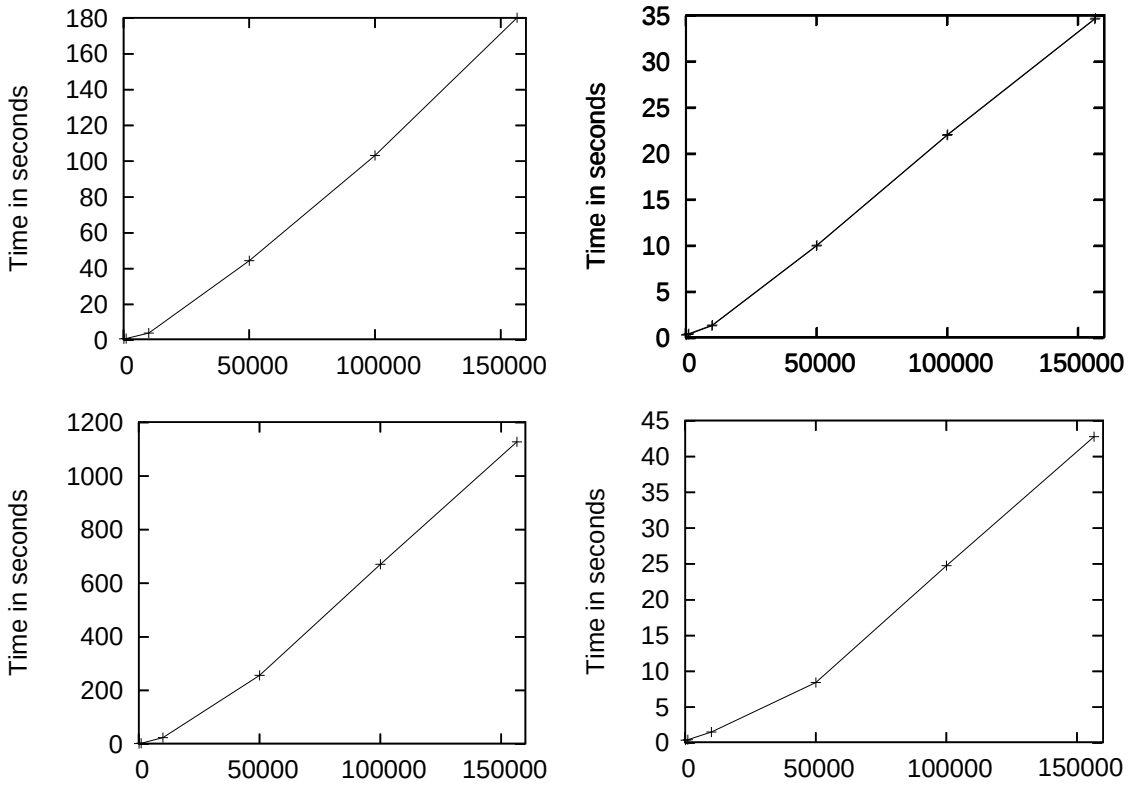

Figure 4.18: *Time required to extract Default&Refine rules for different phonemes from different sized training dictionaries, measured using the FONILEX corpus.*

These trends are further illustrated in Fig. 4.16 4.18 and 4.17. Execution time for alignment, pattern extraction and rule extraction is plotted for a training dictionary as it increases in size. These values were measured on a 1600 MHz Intel Pentium 4 personal computer with 1 GB memory, using the initial *Perl* prototype used during experimentation (System A). In comparison, equivalent algorithms are much faster as implemented in System B, a more robust version of the initial prototype[17], as listed in Table 4.14.

[17]These systems are described in more detail in Chapter 6. System A was developed in *Perl* by the author, and used during algorithm development and experimentation; System B was re-implemented in *Java* (without any algorithmic changes) by members of the CSIR HLT Research Group. The second system was used to build a medium-sized dictionary, as described in Section 6.5.

Table 4.14: *A comparison of computation times in seconds for alignment and Default&Refine rule extraction for two different implementations of the bootstrapping system.*

Task	System A	System B
Alignment: 10,000 words	185.32	8.15
Alignment: 50,000 words	793.23	49.08
Default&Refine: 10,000 words	272.40	26.13
Default&Refine: 50,000 words	1335.36	320.06

4.7.4 ROBUSTNESS TO NOISE

In order to analyse the effect of errors on predictive accuracy, we conduct a number of simulation experiments, using *Afrikaans A*, one of a set of Afrikaans bootstrapped dictionaries, as described in Section 4.3. Based on earlier experience with dictionary developers who are more error prone (see Section 6.3.2.2), we artificially corrupt a fraction of these transcriptions and then measure the predictive accuracy of *Default&Refine* on the corrupted databases.

We introduce two types of corruptions into the transcriptions:

- *Systematic corruptions* reflect the fact that users are prone to making certain transcription errors - for example, in the ARPAbet phone set, *ay* is often used where *ey* is intended. We allow a number of such substitutions, to reflect observed confusions by Afrikaans transcribers.

- *Random corruptions* simulate the less systematic errors that also occur in practice; in our simulations, random insertions, substitutions and deletions of phonemes are introduced.

We generate four corrupted data sets (systematic substitutions and random insertions, substitutions and deletions), where 1%, 2%, 5% and 10% of the words are randomly selected for corruption. We generate *Default&Refine* and *DEC-grow* rule sets with 90% of the words of each (corrupted) dictionary and measure the accuracy of the rules against the remaining 10% (using the original uncorrupted dictionary), and perform 10-fold cross-validation.

The effect of the simulated errors on predictive accuracy is depicted below. In Figure 4.19 the average word accuracy and phoneme accuracy are plotted against the percentage of corrupted words for *DEC-grow* and *Default&Refine*. Note that the most significant effect is due to insertions, as unnecessary insertions cause superfluous graphemic nulls, which introduce alignment errors. This effect is visible for both *DEC-grow* and *Default&Refine*, as both rely on accurate pre-alignments. Figure 4.20 provides a more detailed analysis: the change in average word accuracy and phoneme accuracy is plotted in the same way as above. Here it can be seen that deletions and substitutions affect the predictive accuracy to a similar extent, whether random or systematic. This behaviour is quite different to the behaviour observed later (see Section 6.4), when the position of rules in the extracted rule set is used to predict errors in the training data. As no rules are discarded during standard *Default&Refine*, rule set position does not affect predictive accuracy. Both rule extraction

techniques perform well in the presence of low levels of noise, with *Default&Refine* providing a slight advantage over *DEC-grow*.

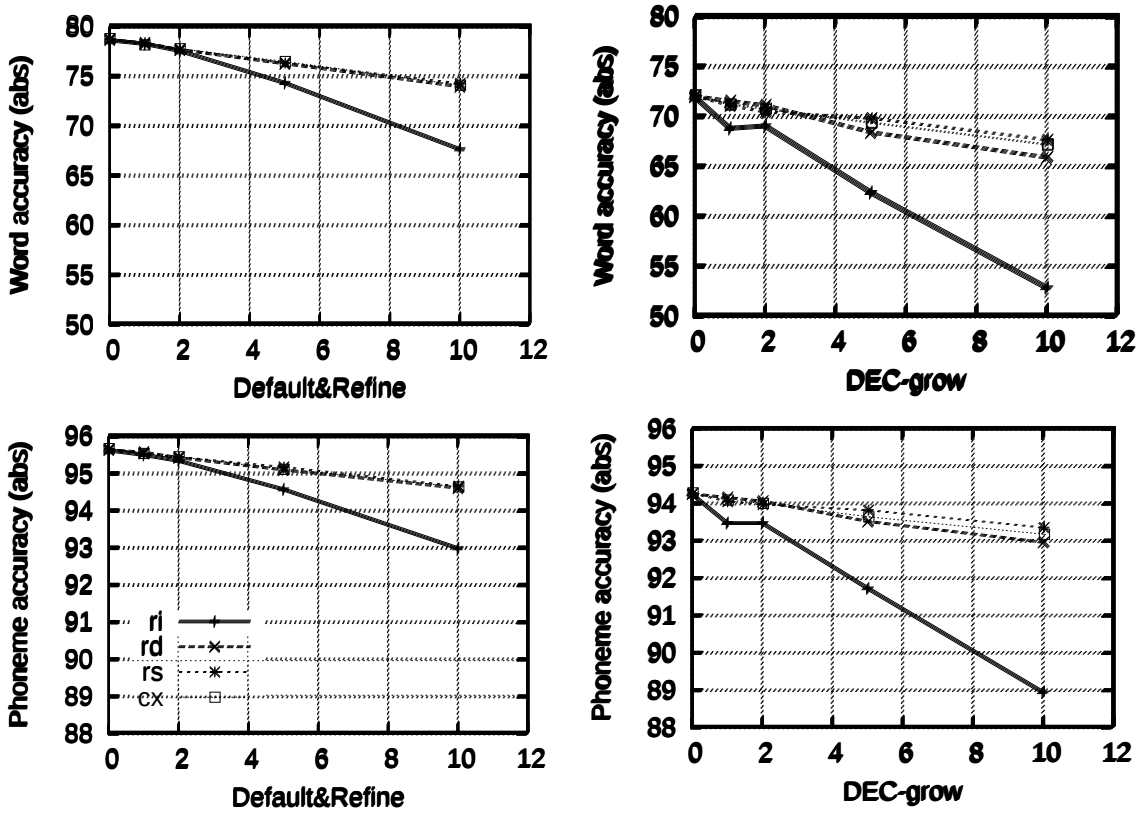

Figure 4.19: *Effect of noise on average phoneme and word accuracy when extracting rules from a corrupted version of the Afrikaans A database. Databases are corrupted with random insertions(ri), random deletions (rd), random substitutions (rs) and systematic substitutions (cx).*

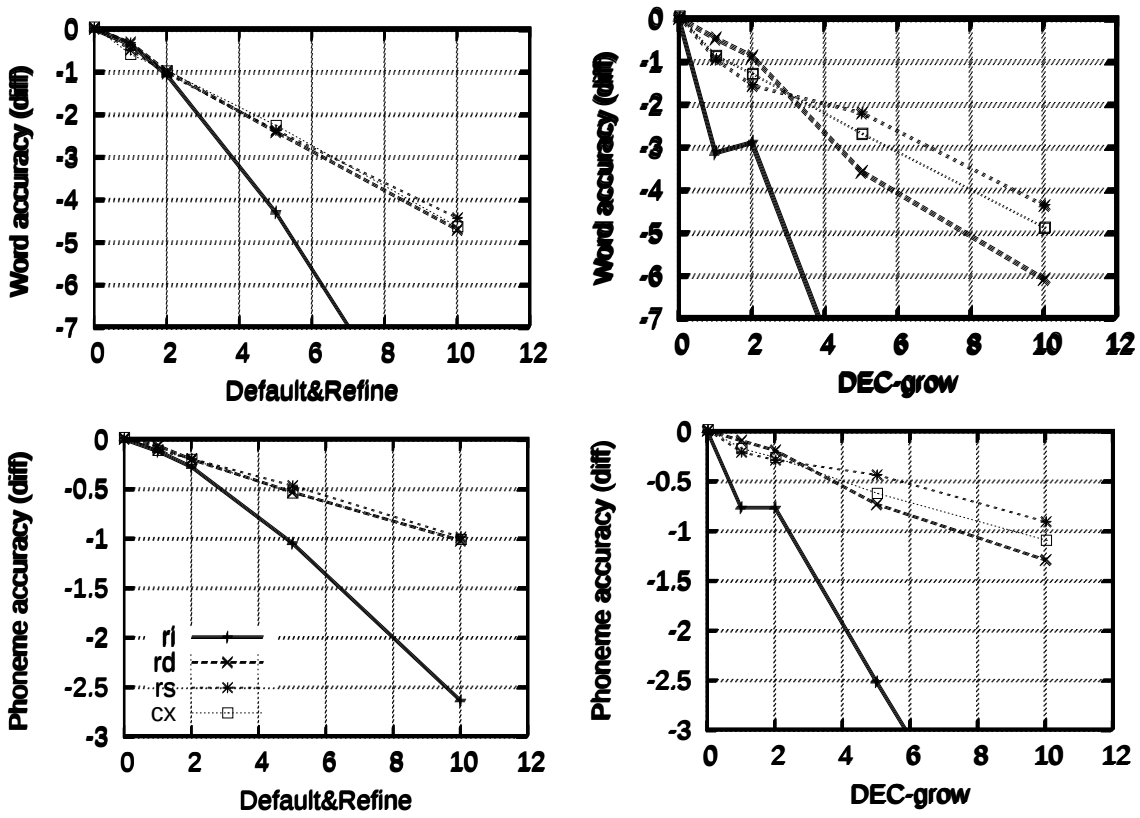

Figure 4.20: *Effect of noise on change in average phoneme and word accuracy when extracting rules from a corrupted version of the Afrikaans A database. Databases are corrupted with random insertions(ri), random deletions (rd), random substitutions (rs) and systematic substitutions (cx).*

4.8 CONCLUSION

In this chapter we analysed the grapheme-to-phoneme conversion task through a set of experiments based on variations of Dynamically Expanding Context (DEC). We proposed an enhancement to the standard approach for grapheme-to-phoneme alignment and defined a new grapheme-to-phoneme conversion algorithm (*Default & Refine*). This algorithm utilises the concept of a default phoneme to extract a cascade of increasingly more specialised rules, and has a number of attractive properties including language independence, rapid learning, good asymptotic accuracy, robustness to noise, and the production of compact rule sets. In subsequent chapters, we utilise both *DEC-min* and *Default&Refine* as grapheme-to-phoneme conversion mechanism during bootstrapping.

Table 4.6 and Figures 4.9 and 4.10 depict an interesting trend: as the rule sets that fully describe the training data become smaller and smaller, the generalisation accuracy of the rule set increases. This raises an interesting theoretical question: What is the smallest possible rule set within a rewrite rule based framework that can fully reconstruct a given set of training data with 100 % accuracy? In the next chapter (Chapter 5) we explore this question further.