An efficient search space representation for large vocabulary continuous speech recognition

Kris Demuynck, Jacques Duchateau, Dirk Van Compernolle * and Patrick Wambacq

November 24, 1999

Katholieke Universiteit Leuven
Department of Electrical Engineering,
Kardinaal Mercierlaan 94,
B-3001 Heverlee, Belgium

Address for correspondence
Kris Demuynck
ESAT - PSI - Speech
Kardinaal Mercierlaan 94
B-3001 Heverlee
BELGIUM

phone: +32 16 321860
fax: +32 16 321986
E-mail: Kris.Demuynck@esat.kuleuven.ac.be

* Lernout & Hauspie Speech Products, Belgium.
Keywords
Continuous speech recognition; Large vocabulary speech recognition; Search algorithms; Context dependent acoustic modelling
Abstract

In pursuance of better performance, current speech recognition systems tend to use more and more complicated models for both the acoustic and the language component. Cross-word context dependent phone models and long-span statistical language models are now widely used. In this paper, we present a memory-efficient search topology that enables the use of such detailed acoustic and language models in a one pass time-synchronous recognition system. Characteristic of our approach is (1) the decoupling of the two basic knowledge sources, namely pronunciation information and language model information, and (2) the representation of pronunciation information - the lexicon in terms of context dependent units - by means of a compact static network. The language model information is incorporated into the search at run-time by means of a slightly modified token-passing algorithm. The decoupling of language model and lexicon allows great flexibility in the choice of language models, while the static lexicon representation avoids the cost of dynamic tree expansion and facilitates the integration of additional pronunciation information such as assimilation rules. Moreover, the network representation results in a compact structure when words have various pronunciations, and due to its construction, it offers partial language model forwarding at no extra cost.

Résumé

A la recherche d’une meilleure performance, les systèmes de reconnaissance de la parole actuelles inclinent à des modèles acoustiques et des modèles de langage de plus en plus compliqués. Des modèles de phones en contexte intramot et des modèles de langage de longue envergure sont maintenant largement répandus. Dans cet article, nous présentons une topologie de recherche qui permet l’utilisation de tels modèles détaillés, dans un système de reconnaissance temps-synchrone à une passe. Caractéristique à notre approche est (1) le découplage des deux sources de connaissance de base, à savoir l’information de prononciation et l’information de modèle de langage, et (2) la représentation compacte d’information de prononciation - le lexique en termes de phones en contexte - au moyen d’un réseau statique. L’information de modèle de langage est incorporée à la recherche au délai d’exécution au moyen d’un algorithme de passage de jeton (token passing) légèrement modifié. Le découplage du modèle de langage et du lexique permet une grande flexibilité dans le choix des modèles de langage, alors que la représentation statique du lexique évite le coût d’expansion dynamique du réseau et facilite l’intégration d’information de prononciation supplémentaire telle que des règles d’assimilation. En plus, la représentation par un réseau est très efficace quand les mots ont des prononciations multiples, et à cause de sa construction, la structure proposée offre l’anticipation partielle de modèle de langage sans coûts supplémentaires.
1 Introduction

Recent developments in large vocabulary speech recognition show the tendency to use ever more complex models like cross-word context dependent models and long-span language models. The resulting complexity however poses a severe problem where computer resources are concerned, especially if these detailed models are to be used in the first and maybe only pass of a recogniser. In this paper we present a memory-efficient search topology that allows the use of both cross-word context dependent models and long-span language models in a single-pass time-synchronous recogniser. The wide range of acoustic models and language models that can be accommodated with the proposed search topology makes it an ideal candidate for research systems. The low memory requirements also form a good starting point for consumer products. To this end however, some extensions may be needed to guarantee real-time recognition.

The remainder of the text is organised as follows. First we briefly depict some concepts concerning search strategies used in large vocabulary speech recognition. Next we indicate some solutions to the above mentioned problem as found in the literature and we present our novel approach. The next two sections describe the two main blocks in our search engine: the lexicon representation and the search algorithm. Next the benefits of our approach are given and some possible extensions are presented. We conclude with an evaluation of our system on the Wall Street Journal benchmark tests.

2 Concepts

The task of the decoding unit in an HMM-based continuous speech recognition system is to find the most likely sequence of words for a given input signal. In large vocabulary speech recognition, the words are typically decomposed into basic sounds called phones. The phones themselves are modelled as a sequence of states. The decoding unit thus has to find the most likely sequence of states given the constraints imposed by the pronunciation dictionary (the order of the phones in the words) and the language model (the order of the words in a sentence).

As the potential search space is tremendous, a good search algorithm is needed to assure that the main part of the search effort is devoted to the most probable sentences. In breadth-first (frame-synchronous) decoders like our system, typically a beam-search algorithm is used for that purpose. To increase the efficiency of the search further, the lexicon is organised in a tree structure: by merging the common prefixes of the words, duplicate work is avoided as much as possible [Bahl et al., 1989, Ney et al., 1992].

Finding the best sequence of phones given the constraints imposed by the lexicon and the language model (LM) can be redefined as finding the best path through a graph that represents all possible phone sequences. A simple conceptual model that illustrates the use of such search graphs for speech recognition is the token-passing paradigm [Young et al., 1989]. One of the basic properties of the token-passing algorithm is that if two or more tokens land in the same node in the graph, only the best scoring token has to be retained (Bellmans’ optimality principle). This way, inferior paths are removed automatically from the search beam.

Applied to continuous speech recognition, this property of the token-passing algorithm allows us to automatically remove alternative word sequences that are guaranteed to be inferior to other word sequences. We therefore need to have a search graph for which alternative paths merge if the possible extensions of the paths are equivalent (identical extension paths and cost functions). Using this principle with a unigram language model leads to a search graph containing a single lexical tree with loop-back links from the word-final nodes to the start node. In the case of a bigram, the cost
functions in the lexical trees depend on the predecessor word, resulting in word-conditioned lexical trees [Ney et al., 1992] as shown at the left side of figure 1.

3 Tractable M-phone N-gram recognition

Recently, considerable effort is put into the use of cross-word context dependent (CD) phone models as they provide better recognition results than word-internal context dependent or context independent (CI) models. The use of cross-word context dependent phone models results however in a tremendous expansion of the lexicon representation. When the lexical tree is augmented with the language model information to form a complete search graph, the structure becomes even bigger. In the literature, different solutions are proposed to cope with this problem.

Progressive (multi pass) search techniques use simplified acoustic and language models in a first recognition pass and record the most interesting parts of the search space in the form of N-best lists [Schwartz and Chow, 1990, Schwartz et al., 1992], or more recently, in the form of word graphs [Oerder and Ney, 1993, Murveit et al., 1993]. The following recognition passes are then confined to the sentences encoded in the word graphs, thus limiting the search space and the memory requirements. For the first recognition pass, the number of cross-word connections and thus the memory requirements can be further limited by exploiting the back-off component of the bigram LM [Austin et al., 1990]. To allow an efficient decoding, this approach requires a minor approximation (extra smoothing) in the bigram LM. This extra smoothing however, has no negative effect on the overall performance of the system [Placeway et al., 1993].

Recent papers show that a static representation of the complete search space is possible if an optimal representation of the search space is chosen. In [Hanazawa et al., 1997] it is shown that for tasks that have large sets of equiprobable words (e.g. street names, ...) in their finite state grammar, a compact representation in the form of directed acyclic word graphs exists. When stochastic N-grams are used, more general optimisation techniques have to be applied to keep the search space representation manageable. In [Antoniol et al., 1992, Federico et al., 1994], the search graph is optimised based on the indistinguishability property of states in a deterministic finite state automaton. In [Riley et al., 1997, Mohri and Riley, 1997, Mohri et al., 1998] all knowledge sources (lexicon, language model and context dependent models) are represented as weighted finite-state transducers, which are combined in an optimal way using algorithms such as transducer composition, transducer determinisation and \( \epsilon \)-removal for weighted automata. This scheme allows the use of cross-word triphones and trigrams in a single pass, even for tasks as complex as the North American Business News task.

An alternative approach to build a recogniser capable of handling complex acoustic and language models in a single pass is to create the search graph dynamically around those search paths that are in the search beam. In [Ney et al., 1992, Beyerlein et al., 1997] lexical trees for each occurring acoustic/language model context are created on demand during the decoding. A more refined dynamic construction of the search space is described in [Odell, 1995]. Advantageous to this approach is that language model and HMMs are highly decoupled, allowing a flexible choice in language models. Disadvantageous are the additional computations required during the recognition. An intermediate solution, allowing a static representation of the pronunciation information, is described in [Austin and Fallside, 1989].

Main purpose of our recognition system is the development and assessment of different prepossessings, acoustic models, language models and other knowledge sources. Flexibility and ease of use is thus our primary concern. A two pass search strategy may mask the benefits of a new or im-
proved knowledge source due to the information loss in between the two recognition stages. A two pass recogniser also requires multiple models to be maintained and needs careful fine tuning of all parameters to assure an optimal combination of both stages. We therefore opted for a single pass recognition system, this despite the fact that multi pass systems tend to be faster than equivalent single pass systems (but speed was only of secondary importance in our system; for an indication of the speed of our system, see table 4). In order to limit the memory requirements and to have minimal constraints on the LM, we use an almost complete decoupling of language model and pronunciation information on the one hand, and a novel compact lexicon-network representation on the other hand [Demuynck et al., 1997]. The information from both basic knowledge sources, language model and lexicon, is integrated on-line during the search by means of a modified token-passing algorithm. Figure 1 illustrates the main differences between the classical approach using search graphs and our approach. The use of the language model as an independent knowledge source, which is integrated on-line with the other knowledge sources is very similar to what is found in stack decoders. Our approach thus bridges to some extent the differences between stack decoders (depth-first) and frame-synchronous (breath-first) decoders.

4 The lexicon representation

4.1 Principles

The introduction of cross-word context dependent phones in a lexicon results in a serious growth of the lexicon representation because all word final and word initial phones have to be replaced with all possible context dependent variants, including the correct arcs to inter-connect them. If however both the word beginnings (prefixes) and the word endings (suffixes) are stored in a tree structure,
the number of word initial and word final phones is reduced to the number of phones instead of the number of words in the lexicon (see figure 2). Consequently, the introduction of cross-word context dependent phones will now only give rise to a limited increase in size of the lexicon representation.

![Figure 2: The number of units needed to describe the cross-word transitions can be strongly reduced if both the word beginnings and the word endings are stored in a tree structure.](image)

Unfortunately, this reasoning is somewhat over-simplified. If both the prefix and the suffix tree are to be perfect, conflicting constraints may arise with every node in the lexicon that must be shared by both the prefix and the suffix tree. Consequently, a compromise is needed. In our system, we opted for a perfect prefix tree as this results in a faster decoding. To limit the effect of the imperfections in the suffix tree, dummy nodes are introduced. These dummy nodes moreover help to reduce the network size if only the last phone of a word can be merged (see section 4.2 for more details). Note that a more compact lexicon representation is possible if compromises are allowed in both suffix and prefix tree. This may however lead to a less efficient decoding.

The word identities have to be stored in the lexicon too, and as this is word specific information, they cannot be stored in the shared prefix or suffix tree. In our system the word identities are stored as labels on the arcs, this way the non shareable mid-part of a word can be as small as a single arc. An example of the resulting network structure is shown in figure 4.

### 4.2 Dummy nodes

Dummy nodes are not related to any acoustic event, i.e. they have no accompanying probability density function. They are used to compact the lexicon network and to mark special situations (e.g. the beginning or end of a word). In our system, three types of dummy nodes are used: START, END and LINK nodes. The END nodes mark the end of the words. The START nodes indicate the beginning of the words, and at the same time they compact the network: from each END node only a single arc is needed to connect the word end through a START node with all matching word beginnings. Figure 3 illustrates this. The LINK nodes are used to compact the network solely: they dispatch all incoming arcs to a set of successor nodes wherever this is beneficial.
Figure 3: START nodes help to reduce the number of cross-word arcs.

Figure 4 shows an example lexicon network. The figure also illustrates two situations in which LINK nodes are beneficial: if only the last phone of a word can be merged and if a word occurs as a whole in the beginning of another word (e.g. inflections of words). This second case is an example of the conflicting constraints imposed by having both a prefix and a suffix tree.

Figure 4: An example lexicon network for right-context dependent phones. The LINK nodes help to reduce the number of arcs in some special cases.

4.3 Properties of the lexicon network

When both the word beginnings and the word endings are merged, a word in the lexicon is described as a shared prefix, a non common mid-part and a shared suffix. The resulting structure is no longer a tree, but a network: nodes can have multiple successors as well as multiple predecessors. The most important properties of the network are:

minimal branching: In order to minimise the search effort during recognition the fan-out of all nodes must be minimal: no node whatsoever may be followed by two or more identical CD phones,
irrespective of the occurrence of intermediate LINK or START nodes. Neither may a node be followed by more than one END node. The minimal branching thus assures that the network is at least as optimal as a lexical tree with respect to the efficiency of the decoding.

directed: All information concerning the words (pronunciation and identity) is confined to the part of the network between the START and END nodes. This part of the network has a left to right topology and is loop free. The loop-back information points from the END nodes to the START nodes and is also loop free. This loop-back information consist of dummy nodes (END, LINK and START nodes) solely.

word tag optimised: As no word specific information can be stored in the shared parts and as the mid-part can be as short as a single arc, the word identity must be stored as a label on one of the arcs in the non common mid-part of the word. To get an optimal compression of the common suffix tree, the word identity is stored on the first non shared arc. The construction of the network (see section 4.5) also guarantees that every path that starts with a START node and ends at an END node contains one and only one word tag. If a pronunciation variant of a word coincides with a pronunciation of another word (homophones), a new super word which refers to both words is introduced. Instead of having to store the identity of both words on two separate arcs pointing to the same node, we now can store the identity of the super word on a single arc. This way the minimal branching property can be maintained, even in the presence of homophones.

4.4 Results of the compression

Table 1 shows the effectiveness of the network representation for a large Dutch lexicon. The lexicon

<table>
<thead>
<tr>
<th>words</th>
<th>object</th>
<th>CI phones</th>
<th>left CD</th>
<th>right CD</th>
<th>triphones</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lexicon Tree</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5k</td>
<td>nodes</td>
<td>13442</td>
<td>14414</td>
<td>198406</td>
<td>/</td>
</tr>
<tr>
<td></td>
<td>arcs</td>
<td>18441</td>
<td>32913</td>
<td>383405</td>
<td>/</td>
</tr>
<tr>
<td>100k</td>
<td>nodes</td>
<td>291358</td>
<td>292711</td>
<td>4491317</td>
<td>/</td>
</tr>
<tr>
<td></td>
<td>arcs</td>
<td>391357</td>
<td>418663</td>
<td>8691316</td>
<td>/</td>
</tr>
<tr>
<td>Lexicon Network</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5k</td>
<td>nodes</td>
<td>5894</td>
<td>8124</td>
<td>9375</td>
<td>36264</td>
</tr>
<tr>
<td></td>
<td>arcs</td>
<td>10834</td>
<td>15612</td>
<td>29056</td>
<td>101446</td>
</tr>
<tr>
<td>100k</td>
<td>nodes</td>
<td>123772</td>
<td>135712</td>
<td>166990</td>
<td>227959</td>
</tr>
<tr>
<td></td>
<td>arcs</td>
<td>222010</td>
<td>237662</td>
<td>468410</td>
<td>675920</td>
</tr>
</tbody>
</table>

Table 1: Number of nodes and arcs needed to represent the 5000 and 100000 most frequent Dutch words.

contains a single pronunciation per word and uses a phone set of size 44. The linear representation consist of 31818 nodes for the 5000 word case and 894261 nodes for the 100000 word case. We assume distinct models for all possible CD phones. In the 5000 word case, 1193 biphones and 23052 triphones actually occur (cross-word or intra-word). For the 100000 word case, these numbers are 1618 and 40176 respectively. As reference, we also give the size of simple lexicon trees, which do use START nodes in order to limit the number of cross-word arcs (see figure 3), but have no compression.
of the suffixes whatsoever. As can be seen in table 1, the introduction of START nodes is only sufficient to cope with the growth of the lexicon representation in the case of left-context dependent phones. For right-context dependent phones or triphones, the suffix tree structure and the LINK nodes are needed.

Table 2 gives more detailed information on the number of nodes and arcs needed in the lexicon representation for the WSJ0 task, using the 918 CD phones that occurred at least 250 times in the SI-84 training sentences. The lexicon contains 10% words with multiple transcriptions. We also give the average fan-in \((f_i)\) and fan-out \((f_o)\) for the different node types. This average fan-in and fan-out can be used to estimate the impact of a dummy node. For example, the total number of arcs in the network representation doubles if no LINK nodes are used: removing all LINK nodes requires the \(N \times (f_i + f_o)\) arcs going to or starting from the LINK nodes to be replaced by \(N \times (f_i \times f_o)\) arcs to preserve all possible paths through the network (\(N\) is the number of LINK nodes).

<table>
<thead>
<tr>
<th>node-type</th>
<th>5k-closed</th>
<th>20k-open</th>
<th>64k</th>
</tr>
</thead>
<tbody>
<tr>
<td>normal</td>
<td>11344</td>
<td>32927</td>
<td>88732</td>
</tr>
<tr>
<td>START</td>
<td>213</td>
<td>224</td>
<td>242</td>
</tr>
<tr>
<td>END</td>
<td>22.0–9.4</td>
<td>40.0–9.9</td>
<td>62.8–10.1</td>
</tr>
<tr>
<td>LINK</td>
<td>659</td>
<td>1652</td>
<td>4362</td>
</tr>
<tr>
<td>arcs</td>
<td>41125</td>
<td>112363</td>
<td>303838</td>
</tr>
</tbody>
</table>

Table 2: Size of the lexicon network using 918 triphones for the 5k, 20k and 64k non verbalised punctuation word sets in the WSJ0 task. The second line contains the average number of input \(\rightarrow\) output arcs for the given type of node.

**4.5 Constructing the network**

**4.5.1 The main routine**

To construct the lexicon network the following steps are taken:

1. Make a context independent lexicon tree. All words are added to a standard tree structure with their word identity moved to the first non shared arc. Homophones are treated as a single ‘super’ word (see section 4.3).

2. Connect all word final nodes with an END node, connect the END node with the START node and add an extra ROOT node in front of the START node. The ROOT node collects arcs to all START nodes, allowing easy access to the START nodes at all time.

3. Merge the word endings (make the ‘inverse’ suffix tree) with a recursive routine. Starting from the ROOT node, the recursive merge routine first scans for successor nodes, until an END node is reached. Nodes are then merged on the return path (going from the END nodes up to the START nodes). Note that the inverse tree is in general far from perfect: two sister nodes (nodes with a common successor node) that represent the same phone can only be merged if they have
all their successor nodes in common and if the involved arcs (the arcs to the successor nodes) contain the same word labels (or no word label). This step is not necessary, but will speed up the conversion to a context dependent description later on.

4. Mark all nodes as being context independent and start the recursive routine that converts the network to a context dependent description starting with the ROOT node. This stage also introduces the LINK nodes in the network to limit its size. The procedure that converts phones from a context independent to a context dependent description is given in section 4.5.2.

5. Clean up LINK nodes that have no effect \((f_i + f_o \geq f_i \times f_o)\).

4.5.2 Converting the network to a context dependent description.

This routine and the underlying split routine require that the input network complies with the conditions given in section 4.3. If so, both routines will result in a lexicon network that still satisfies these conditions. The routines can handle word-internal as well as cross-word phones. In the description of the algorithms, following naming conventions are followed: the central node is the node for which a context dependent variant is to be created, while the current node is the node in the context of the central phone or the central phone itself that is currently being handled by the split routine. Sister nodes are nodes which have at least one common successor.

1. Search for a context dependent phone that matches the context of the central node (check if both a left and a right context can be found which match the phone description, this check must also be able to look across word boundaries).

2. Make duplicates of the central node and/or some nodes in the context to take care of contexts that do not match the chosen CD phone. This is done by the recursive split routine described in section 4.5.3. The split routine is called successively for both the left and the right context. The central phone is thus handled twice, once for the left context and once for the right context.

3. Do the same for all successor nodes that are not yet handled and are not an END node (recursively). Note that whenever a successor node is handled, new successor nodes may have been created by the split routine (section 4.5.3). This way, what was originally one CI node is replaced with multiple CD nodes.

4. Mark the current node as being context dependent, and change the original context independent phone label to the correct context dependent phone label.

5. Try to merge the newly created CD unit:
   If the central node has only one successor
   
   then try to merge with an equivalent sister-node, i.e. a sister node which represents the same CD phone, which has also one successor, and which has an identical label on the outgoing arc (or also no label).

   else group the outgoing arcs per label (the arcs with no label also form a group) and insert a LINK node in front of each group. Next, scan for equivalent LINK nodes (i.e. LINK nodes having the same group of successor nodes) and merge them.

In our current implementation LINK nodes are only inserted if there is a perfect match between two sets of successor nodes, as specified by the ‘else’ statement in step 5 (merging the newly created CD
unit). We expect still a substantial decrease in the number of arcs if LINK nodes are also inserted before large common subsets of successor nodes, without the need of having a perfect match between the two sets of successor nodes.

4.5.3 The split routine

This routine isolates matching from non-matching contexts for a given central node (i.e. the node that is to be changed to a context dependent variant). The routine is called twice: first to handle the left context, next to handle the right context. The working of the routine is depicted in figure 5.

First, all matching predecessor nodes (left context) or matching successor nodes (right context) are handled in a recursive way. This way, the routine will visit all nodes in the left/right context that match the given context description. By handling the predecessor/successor nodes first, we assure that the contexts are isolated starting from the most distant nodes in the given context and then back to and including the central node. In other words, the split routine first traverses the network to reach
the most distant nodes and then isolates the matching contexts from the non matching contexts on the return path.

The context may span more than one node if long-span contexts are used (e.g. quinphones), or if dummy nodes are encountered (e.g. when a word boundary is crossed). In these cases, the decision whether a predecessor or successor node is part of the matching context has to be delayed until all possible continuations are scanned. For example, whether a LINK node is part of a matching context or not always depends on the LINK node’s predecessors/successors. This is however not a problem as contexts are isolated on the return path, i.e. after all continuations are investigated. By the time the actual splitting has to be done, the status of all predecessor/successor nodes is known.

In order to isolate the matching contexts, duplicates of the current node (the node currently reached by the recursive split routine) are made. As can be seen in figure 5, there are three distinct cases to be considered. The complex case for the right context occurs if the following three conditions are fulfilled: the current node is not a START nor a LINK node, the current node is not the central node and the current node has more than one predecessor. Whenever the split routine creates a new START node, an extra arc to the ROOT node has to be added too (see step 2 in section 4.5.1).

Figure 6 demonstrates the above described operations for quinphone \([c][ab]-x-[d][e]\). First the left context is handled, starting with \([x]\) as current phone. The first phone seen in the left context of \([x]\) is \([a]\) and since \([a]\) is not a START node, the left context is isolated as depicted in Figure 6.

### Starting point:

- **c**
- **a**
- **p**
- **d**
- **e**
- **q**
- **b**
- **r**
- **s**

matching CD-phone: \([c][ab]-x-[d][e]\)

### Isolate left context:

1.1.1
- **c**
- **a**
- **p**
- **d**
- **e**

1.2.1
- **q**
- **b**
- **r**
- **s**

2.
- **x**

### Isolate right context:

2.1.1
- **p**
- **d**
- **e**

2.1.2
- **q**
- **b**
- **x**
- **d**
- **s**

2.2
- **p**
- **d**
- **e**

Figure 6: An example of how contexts are isolated by the split routine. The dashed lines indicate the path followed by the recursive split routine, while the numbers give a logical index to each instance of the routine. The gray shaded nodes are the newly created duplicates.

Figure 6 demonstrates the above described operations for quinphone \([c][ab]-x-[d][e]\). First the left context is handled, starting with \([x]\) as current phone. The first phone seen in the left context of \([x]\)
is [a]. As [a] matches the context description so far, a new instance of the recursive split routine is started (instance number 1.1 on the figure) with [a] as current phone. Next the left context of [a] is scanned, with [c] being the only predecessor. As [c] matches, a new instance of the routine is started. By now, the end of the context description is reached so nothing has to be done but signalling a match by returning the correct value to the calling routine. Back in [a] (instance 1.1), all predecessors are handled. As all predecessors match, the split action as depicted in figure 5 degenerates and nothing is done, except for returning a match to the caller (instance 1 of the split routine). Next context [b] is scanned, and the whole story is repeated, except now a mismatch is returned. Back in [x], both predecessors are now handled with one match and one mismatch. The correct context is now isolated by creating a duplicate of the current phone (the gray shaded node). Next the right context is handled, again starting with [x] as current phone. Node [d] is split according to the complex case of figure 5 and thus two new duplicates are created. The first duplicate is connected with the correct context (the node from which instance 2.1 is instantiated), the second duplicate is connected with the remaining predecessor nodes. Back in [x], the newly created duplicate does not need to be scanned as this is a guaranteed mismatch. Scanning these new duplicates can be avoided by first counting the number of outgoing arcs and then handling only that many arcs. New arcs then of course always have to be added at the end of the list of arcs. As a last step, the correct context of [x] is isolated. The resulting network still encodes the same phone sequences, but the requested context is now isolated.

5 The search algorithm

As the two knowledge sources that make up the search graph (the pronunciation information and the language model) are separated in our approach, a somewhat modified beam-search algorithm is needed. Basically, we still use a time-synchronous beam-search algorithm based on the token-passing paradigm [Young et al., 1989, Odell, 1995]. The tokens still accumulate the observation scores and collect the trace-back information, but instead of referring directly to a point in the search graph, references to both knowledge sources are needed.

5.1 Conditioned token recombination

For the integral search graph approach (see figure 1), it is sufficient to know the position of the tokens in the search graph to determine whether two tokens are equivalent: if two tokens reside in the same node in the search graph then they are equivalent and only the best one has to be retained (recombination of tokens). In our approach, tokens contain multiple references, so the position of the token in the lexicon network in itself is not sufficient: tokens can be recombined only if all references are identical. In our system, tokens contain the following references:

The lexicon node gives the location of the token in the lexicon network, thus making the link between token and pronunciation information.

The word identity. The word identity equals ‘unknown’ at the beginning of the lexicon network and is filled in whenever a labelled arc (see section 4.3) is traversed. This reference in fact makes that the tokens traverse the lexicon network as if it was a lexical tree: recombination of two tokens at the same position in the suffix tree of the lexicon network is only possible if they have the same word identity.

The LM context incorporates the language model into the search. The LM context determines uniquely the impact of the LM on the score. For a word-based bigram this is the predecessor
word, for a finite state grammar it is the current state and for long-span LMs it is the word context that can still have an effect on the LM score. This reference makes that the tokens traverse the lexicon network as if there were distinct tree copies for each LM context.

5.2 The search routine

The search algorithm is straightforward. For each input frame, we first mark all tokens in the search beam as being old, this to avoid recombination between the old tokens and the newly created tokens which have observed one additional frame of acoustic data (see section 5.3 for more details on how this is done). Next, the acoustic scores in the tokens are augmented with the new observation scores, and copies of the tokens are propagated to all successor nodes as indicated by the lexicon-network. If one or more of the successor nodes are LINK nodes, the tokens should be propagated further until a non LINK node is reached. If a labelled arc is traversed, the word identity is stored in the token and the LM score is updated. If two or more of these newly created tokens end up with the same set of references, only the best one has to be retained (recombination of tokens). Finally, all tokens that have landed in an END node are handled: the LM contexts are updated, i.e. the old context is augmented with the new word, and the tokens are propagated further across the word boundaries.

For what concerns the super words in the lexicon network (see section 4.3): these super words were originally introduced to satisfy the word tag optimised constraint. For the decoding, each arc referring to a super word can be regarded as the corresponding set of arcs labelled with the different homophones that make up the super word. According to this approach a token that traverses such an arc gives rise to multiple new tokens, i.e. one for each homophone in the set. We however opted for a different approach. When a token traverses a labelled arc which refers to a super word, multiple LM queries are performed to find the best LM score over all homophones in the super word. The optimal LM score then serves as the LM score of the super word. This way only one token is needed to handle a complete set of homophones. This trick however can only be kept up as long as no END node is encountered, this because the LM is unaware of super words and thus cannot expand an existing LM context with a super word. Tokens referring to a super word thus need to be broken up into separate nodes - one for each of the individual homophones - when they land in an END node.

5.3 Efficient recombination

Recombination in the integral search graph approach (see figure 1) is straightforward: if two tokens end up at the same place in the graph, only the best one has to be retained. When multiple references are used, an efficient scheme is needed to check whether tokens have an identical set of references. We implemented and compared two schemes:

A hash table: To find possible candidates for recombination, we combine all references of the token into one index in a hash table. Only the tokens in that slot of the hash table are possible candidates for recombination and have to be checked (comparison of all references). For optimal performance, the size of the hash table has to be somewhat larger than the maximum beam size (number of tokens active for a given frame). Recombination between old and new tokens (see section 5.2) is avoided by clearing the hash table at the beginning of each new frame: as only the new tokens are in the hash table, the recombination is limited to the new tokens only.

Sorted lists: The tokens are organised as sorted lists attached to the nodes of the lexicon network. To avoid recombination between old and new tokens, both an old and a new list of tokens are needed. During the search, a complete list of old tokens is propagated to a next node in the
lexicon network, where they will be added to a new list of tokens. If this node has a non empty list of new nodes, a new combined sorted list is made and all equivalent tokens are recombined. The use of sorted lists minimises the number of comparisons needed for this operation.

Both methods showed to perform equally well, except for tokens that have to be moved across word boundaries. Moving tokens across word boundaries is rather complex as at least two levels of dummy nodes (END and START nodes) have to be traversed, both having a large fan-out (see table 2). Collecting the tokens in a list, and moving the complete list of tokens instead of the individual tokens across the word boundary reduces the overhead, and was therefore preferred.

Another important issue is the order in which the tokens are expanded. As mentioned in section 4.3, the inner part of the lexicon network, i.e. the part between the START and the END nodes, is loop free. This allows the definition of a node ranking: a node A is said to have a more advanced position in the lexicon network than a node B if node A can be reached from node B. If the tokens in the search beam are handled starting with the tokens that have the most advanced position in the lexicon and so back to the least advanced ones, then tokens will either propagate to positions in the lexicon network that are already taken due to the self transitions in the HMMs or to new unoccupied positions. This way, excessive token creation and destruction can be avoided as for most of the transitions an equivalent token will be available for recombination. In case of recombination, no new token has to be created: if the new transition results in a better score then the already present equivalent token is overwritten with the new information, otherwise the already present token is retained. As described earlier, the tokens in our system are organised in small lists attached to the active lexicon nodes. These active lexicon nodes are again linked together in one large list, allowing us to handle all active nodes without having to run through the complete lexicon network to locate them. By adding newly activated lexicon nodes at the front of this list, and by organising the arcs in the lexicon network so that the transitions to the most advanced lexicon positions are the first to be executed and the self transition the last, an almost optimal order of the active lexicon nodes is assured.

The efficiency of the decoder does not only depend on the algorithm used to find identical sets of references, also the optimality of the references matters. The references to the lexicon node and to the word identity are guaranteed to be optimal by the lexicon construction (minimal branching and word tag optimised). The LM contexts have to determine uniquely the impact of the LM on the score, but for optimality we also need a minimal set of LM contexts. In the case of N-grams the N-1 predecessor words are sufficient to determine the impact of the LM, but they are in general not minimal. If the set of N-1 predecessor words is an unseen context, i.e. the context was never observed in the training data of the LM, then the LM context can be limited to the largest set of M predecessor words that actually occurred \((M < N - 1)\). For example, if a trigram doesn’t have entries of the form “THIS IS <word>” nor does it have entries of the form “THAT IS <word>” (both “THIS IS” and “THAT IS” are unseen contexts), then the context can be limited to “IS”, this way relaxing the recombination constraints. This modification resulted in a 12% reduction of the search effort (average number of tokens in the search beam) for the WSJ Nov92 20k-nvp open vocabulary task using the standard trigram.

### 6 Benefits and extensions

The new search topology reduces the memory requirements drastically: the compact lexicon representation on its own reduces the memory requirements by a factor of 20 and higher; the impact of the decoupling strongly depends on the language model that is used. As memory is no longer an issue, we are able to incorporate cross-word triphones or quinphones and long span N-grams or vary-grams into...
the first decoding pass. Time requirements may however still impose constraints on the complexity of
the models that can be used.

Next to the memory reduction, the proposed search topology also offers some other benefits:

- The interface with the language model, as described in section 5.2, is limited to two actions:
queries about the probability of a new word given the LM context, and the creation of new
contexts given an old context and a new word. This decoupling between language model and
all other knowledge sources allows a great flexibility in the choice of language models.

- As the word identity is known right after the first arc non common with other words is traversed,
the LM can be applied before the end phone of a word is reached. This offers some form
of (partial) LM forwarding. In the common prefix part of the network (see section 4.1), the
unigram probabilities can be used as a first estimate of the LM cost [Steinbiss et al., 1994], this
way avoiding abrupt changes in the scores when labelled arcs are encountered.

- Other pronunciation information such as cross-word assimilation rules can be pre-compiled in
the lexicon network using similar algorithms as described in section 4.5, and except for the
larger search space when introducing new pronunciation variants, no additional search cost is
introduced.

- Alternative pronunciations are handled most efficiently by the network structure. Contrary to
the tree structure, multiple local alternatives (e.g. USES: Y UW [S/Z][AH/IH] Z) do not result in
an exponential growth of the lexicon representation.

The lexicon network can still be improved when state tying is used. The network described so far
is optimal on the phone level: no node whatsoever is followed by two or more identical CD phones.
Every node however still has to be replaced by the correct sequence of states, and if state tying is
used, several of these CD phones may start with one or more identical states. Optimality of the
lexicon network on the state level is thus not guaranteed. The optimisation of the lexicon network on
the state level reduced the search effort (average number of tokens in the search beam) with 40%. The
same improvement was observed for both a small number of tied states (2000) and a large number of
tied states (10000).

The described search topology was also extended to produce word graphs in the framework of a
multi pass recognition system.

7 Evaluation on the WSJ-task

As to allow comparison with other systems, we evaluated our system on the ARPA Wall Street Journal
November 92 benchmark test. For all tests, the single pass time-synchronous system as described in
this paper, was used.

7.1 The models

The acoustic models were trained on the standard WSJ0+WSJ1 SI-284 training data (37516 sentences
from 284 speakers). For the input features, 12 Mel frequency cepstral coefficients and the logarith-
m of the energy (which replaces the zeroth cepstral coefficient) are calculated every 10ms on a 30ms
frame. These 13 parameters are then mean-normalised and augmented with their first and second order
time derivatives, resulting in 39 parameters in total. To decrease the correlation between the param-
eters, an extra linear transformation is applied on the final 39 parameters [Demuynck et al., 1998].
To cope with the variable amount of noise in the recordings, the leading and trailing noise segments in the sentences are limited to approximately 200ms by means of a simple speech detector [Van Gerven and Xie, 1997]. The cepstral mean is calculated over the speech frames only, disregarding leading, trailing and intermediate noise segments.

The context dependent phones are modelled as 3-state left-to-right first order Hidden Markov Models. The observation functions in the states are modelled as a mixture of gaussians. The system uses 20254 gaussians in total, which are to a large extent shared amongst the different states using our ‘reduced’ semi continuous modelling [Demuynck et al., 1996, Duchateau et al., 1998]. For all evaluations, our FRoG-system (Fast Removal of Gaussians) [Demuynck et al., 1996, Duchateau et al., 1998] is used to reduce the computational load imposed by the gaussians. As a result, only 1 out of 19 gaussians on average needs to be evaluated during the recognition.

The states themselves are tied over the different context dependent phones using a decision tree approach. In total 10436 states are modelled resulting in 33169 distinct cross-word position independent triphones. No gender-dependent modelling or function-word modelling is used, nor did we use cross-word assimilation rules.

The language models used, are the standard (non verbalised pronunciation) WSJ language models trained on the 37M word normalised training text material. To limit the memory needed by the LM, both the word identity and the log probability are stored as 16 bit quantities. The LM information is stored in a tree structure as shown at the right side of figure 1. The discount information is added to the same tree.

The amount of memory required by the different knowledge sources is given in table 3. The lexicon network is transformed to the state level (3 states per phone), and then further optimised as mentioned in section 6. The resulting network contains 102902 nodes and 191150 arcs for the 5k word set, and 215843 nodes and 414159 arcs for the 20k word set. Building and optimising the lexicon network takes about 10 seconds for the 5k word set and 40 seconds for the 20k word set on a Pentium-II 450Mhz processor.

<table>
<thead>
<tr>
<th></th>
<th>acoustic models</th>
<th>FRoG network</th>
<th>lexicon network</th>
<th>language model</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>5k bigram</td>
<td>13.0MB</td>
<td>6.5MB</td>
<td>2.0MB</td>
<td>3.4MB</td>
<td>24.9MB</td>
</tr>
<tr>
<td>5k trigram</td>
<td>13.0MB</td>
<td>6.5MB</td>
<td>2.0MB</td>
<td>25.7MB</td>
<td>47.1MB</td>
</tr>
<tr>
<td>20k bigram</td>
<td>13.0MB</td>
<td>6.5MB</td>
<td>4.4MB</td>
<td>6.2MB</td>
<td>30.1MB</td>
</tr>
<tr>
<td>20k trigram</td>
<td>13.0MB</td>
<td>6.5MB</td>
<td>4.4MB</td>
<td>41.7MB</td>
<td>56.6MB</td>
</tr>
</tbody>
</table>

Table 3: The memory required by the different knowledge sources.

7.2 Results

In a first set of preliminary experiments on the 20k-open-bigram task, the effect of the partial LM forwarding was investigated. The acoustic models used for these preliminary experiments did not yet have the decorrelation matrix at the end of the preprocessing, and thus show somewhat worse results than the experiments reported further on. In both cases, the maximum of the uni-gram probabilities of all possible word continuations is used as an early estimate of the real LM cost (uni-gram forwarding). As can be seen in figure 7, enabling the partial LM forwarding decreases the search effort (average number of tokens in the beam) by 30%. The overhead of the extra LM queries needed for the LM
forwarding is kept small by introducing an LM cache system. The cache system is implemented as a simple hash table which maps LM contexts and word identities to the corresponding log probability.

Table 4 shows the recognition effort (time and memory) and the results obtained with our recognition system on the November 92 tasks. The timings were obtained on small but representative subsets of the test-data, containing one sentence per speaker and thus eight sentences in total. The real time factor is the number of seconds needed to process one second of speech on a Pentium-II 450Mhz processor (SpecInt’95: 18.9, SpecFp’95: 13.3). As our preprocessing removes leading and trailing silence, the number of silence frames in the statistics is minimal. By making the size of the LM cache proportional to the expected number of tokens in the search beam, an almost constant hit-rate can be achieved irrespective of search-effort and language model. For the large cache tables used in table 4, the hit-rate is about 96%. Making the LM cache ten times smaller decreases the hit-rate to 94.5%. The LM queries that are not intercepted by the LM cache add only 2% to 4% to the search-time. Replacing the stochastic N-grams with more complicated language models is thus a viable option. Tables 3 and 4 also clearly show the memory efficiency of the proposed system. The static part of the search structures (the lexicon network) as well as the dynamic part of the search structures (the tokens and the back-trace information) are small compared to the language model and the acoustic model.

More detailed information on the search effort needed for the different test-sets and on the token distributions is given in table 5 and in figure 8. The detailed statistics were obtained on a small part of the test-data (the same subsets as used for the timing information). Tokens in our system are instantiated based on all references: tokens in the same lexicon node with a different LM context and/or word identity are not recombined, and thus lead to multiple tokens per lexicon node. As described in section 5.3 these multiple tokens are stored in sorted lists attached to the lexicon nodes. Figure 8 shows the distribution of the length of these lists for the 20k-word trigram case. Although the average number of tokens per lexicon node is only 1.8, large values (up to 20 and more) do occur, thereby justifying the need of having an efficient organisation of the tokens (sorted lists) for fast recombination. Table 5 also shows the effects of replacing a bigram with a trigram: (1) As expected,
Table 4: Recognition effort and result obtained on the WSJ-nov92 5k closed vocabulary and 20k open vocabulary evaluation tasks (non verbalised punctuation).

<table>
<thead>
<tr>
<th>beam-width (#tokens)</th>
<th>acoustic models</th>
<th>search</th>
<th>LM cache</th>
<th>LM tokens</th>
<th>LM other</th>
<th>result (WER)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>average</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5k bigram</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2039 6000</td>
<td>0.98</td>
<td>0.33</td>
<td>0.01</td>
<td>0.01</td>
<td>0.40M</td>
<td>0.46M 0.03M 5.34%</td>
</tr>
<tr>
<td>3318 12000</td>
<td>1.01</td>
<td>0.55</td>
<td>0.02</td>
<td>0.01</td>
<td>0.80M</td>
<td>0.85M 0.06M 4.84%</td>
</tr>
<tr>
<td>5236 24000</td>
<td>1.04</td>
<td>0.90</td>
<td>0.03</td>
<td>0.02</td>
<td>1.60M</td>
<td>1.48M 0.11M 4.56%</td>
</tr>
<tr>
<td>8163 48000</td>
<td>1.08</td>
<td>1.39</td>
<td>0.06</td>
<td>0.02</td>
<td>3.20M</td>
<td>2.49M 0.16M 4.45%</td>
</tr>
<tr>
<td><strong>maximum</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1798 6000</td>
<td>0.97</td>
<td>0.28</td>
<td>0.01</td>
<td>0.01</td>
<td>0.40M</td>
<td>0.45M 0.04M 3.68%</td>
</tr>
<tr>
<td>2943 12000</td>
<td>1.00</td>
<td>0.48</td>
<td>0.02</td>
<td>0.02</td>
<td>0.80M</td>
<td>0.77M 0.05M 3.10%</td>
</tr>
<tr>
<td>4745 24000</td>
<td>1.03</td>
<td>0.78</td>
<td>0.04</td>
<td>0.03</td>
<td>1.60M</td>
<td>1.30M 0.08M 2.91%</td>
</tr>
<tr>
<td>7550 48000</td>
<td>1.06</td>
<td>1.25</td>
<td>0.06</td>
<td>0.04</td>
<td>3.20M</td>
<td>2.20M 0.15M 2.84%</td>
</tr>
<tr>
<td>3039 12000</td>
<td>0.94</td>
<td>0.50</td>
<td>0.02</td>
<td>0.02</td>
<td>0.80M</td>
<td>0.81M 0.04M 11.09%</td>
</tr>
<tr>
<td>5161 24000</td>
<td>0.97</td>
<td>0.86</td>
<td>0.04</td>
<td>0.02</td>
<td>1.60M</td>
<td>1.44M 0.08M 10.60%</td>
</tr>
<tr>
<td>8614 48000</td>
<td>1.01</td>
<td>1.46</td>
<td>0.06</td>
<td>0.03</td>
<td>3.20M</td>
<td>2.40M 0.12M 10.49%</td>
</tr>
<tr>
<td>14152 96000</td>
<td>1.06</td>
<td>2.42</td>
<td>0.10</td>
<td>0.06</td>
<td>6.40M</td>
<td>4.46M 0.18M 10.40%</td>
</tr>
<tr>
<td><strong>20k bigram</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2733 12000</td>
<td>0.92</td>
<td>0.46</td>
<td>0.02</td>
<td>0.02</td>
<td>0.80M</td>
<td>0.76M 0.04M 9.98%</td>
</tr>
<tr>
<td>4639 24000</td>
<td>0.94</td>
<td>0.78</td>
<td>0.04</td>
<td>0.04</td>
<td>1.60M</td>
<td>1.28M 0.08M 9.20%</td>
</tr>
<tr>
<td>7754 48000</td>
<td>0.99</td>
<td>1.32</td>
<td>0.06</td>
<td>0.05</td>
<td>3.20M</td>
<td>2.25M 0.13M 8.65%</td>
</tr>
<tr>
<td>12831 96000</td>
<td>1.03</td>
<td>2.17</td>
<td>0.12</td>
<td>0.09</td>
<td>6.40M</td>
<td>4.27M 0.27M 8.61%</td>
</tr>
<tr>
<td><strong>20k trigram</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20 k trigram</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5k bigram</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5k trigram</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20k bigram</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20k trigram</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5: Some detailed statistics about the search beam (average per frame).

<table>
<thead>
<tr>
<th>average number of . . .</th>
<th>5k bigram</th>
<th>5k trigram</th>
<th>20k bigram</th>
<th>20k trigram</th>
</tr>
</thead>
<tbody>
<tr>
<td>tokens in the search beam</td>
<td>5234.1</td>
<td>4750.1</td>
<td>8574.5</td>
<td>7787.3</td>
</tr>
<tr>
<td>active LM contexts</td>
<td>10.9</td>
<td>11.6</td>
<td>15.2</td>
<td>17.7</td>
</tr>
<tr>
<td>active lexicon nodes</td>
<td>3293.5</td>
<td>2874.8</td>
<td>4830.7</td>
<td>4091.7</td>
</tr>
<tr>
<td>tokens per active lexicon node</td>
<td>1.6</td>
<td>1.7</td>
<td>1.8</td>
<td>1.9</td>
</tr>
<tr>
<td>tokens per active LM context</td>
<td>479.1</td>
<td>411.4</td>
<td>564.4</td>
<td>441.1</td>
</tr>
<tr>
<td>LM cache queries</td>
<td>1459.1</td>
<td>1425.6</td>
<td>2460.3</td>
<td>2737.9</td>
</tr>
</tbody>
</table>

the number of LM contexts increases. The increase however is very limited. (2) The number of tokens per LM context decreases. The trigram is a more powerful language model than the bigram, and thus reduces the perplexity (uncertainty). This reduction in perplexity results, by means of the partial LM forwarding, in a decrease in the search effort. The combination of both effects makes that the overall search effort for the trigram case decreases.
8 Conclusions

In this article, a memory efficient search topology that is able to handle all available knowledge sources (intra- and cross-word context dependent phones, long-span language models, assimilation rules, ...) in a single time-synchronous recognition pass, is described. The decoder was originally designed for system development work and thus imposes only very few constraints on the different knowledge sources. The pronunciation information is pre-compiled in a pronunciation network which is then converted to a context dependent description. A novel compact lexicon-network representation counteracts the network growth when using cross-word context dependent phones. The language model acts as an independent knowledge source which is consulted at run-time by the search engine through a unified interface. The resulting topology is not only flexible in use, it is also very memory efficient and all of this at a competitive decoding speed. Evaluations on the Wall Street Journal benchmark tests already showed that the use of more complex language models (trigrams instead of a bigrams) results in a faster recognition, despite the increased complexity of the decoding.

Acknowledgement

This research was supported by IWT Research Contract 940044, entitled "Continue spraakherkenning van grote vocabularia in het Nederlands", from the Flemish government.
References


