A method for efficient storage and rapid application of context-sensitive phonological rules for automatic speech recognition

by Robert L. Mercer
Paul S. Cohen

In an automatic speech-recognition system, the application of phonological rules to phonemic strings in order to create phonetic graphs is a computationally time- and storage-consuming process. A great many such graphs must be constructed during the decoding phase; thus it is important to be able to rapidly construct phonetic graphs for strings of words from graphs of individual words. However, because many phonological rules operate across word boundaries or require interword context, it is not possible to determine a unique, context-independent phonetic graph for a word. We describe a method for determining the phonetic graph for a word in isolation, together with auxiliary information to allow phonetic graphs for different words to be rapidly interconnected to form a phonetic graph for a string of words; the method also reduces storage requirements significantly.

Introduction

Phonological rules are used in systems for speech recognition by computer in order to generate the many and varied pronunciations ("phonetic realizations") of underlying, somewhat idealized pronunciations ("phonemic baseforms"). In addition to normal inherent variation, the range of pronunciations of words considered in isolation is caused, *inter alia*, by differences in social and geographical dialect, idiolect, style, and pace. In the context of other words ("continuous speech"), further pronunciation adjustments occur. The possible phonetic realizations for a given word can be represented conveniently and compactly in the form of directed graphs, which also permit automatically

determinable probabilities to be associated with different pronunciations [1].

References [2–4] describe the first use of phonological rules and graphs in a system for the automatic recognition of continuous speech. Reference [5] gives a detailed description of a flexible module (implemented in an experimental speech-recognition system [6]) for generating phonological graphs, as well as a comprehensive set of rules for American English.

Much of the very considerable computational time and space needed for generating those graphs was devoted to the production of contextual variants. We have devised methods to reduce both processing time and storage requirements. These methods take advantage of a novel, computationally efficient representation of the possible phonetic realizations of all the words of a particular lexicon as being sequences of subgraphs—from a small set of such subgraphs—which are precomputed and stored. The words are specified in such a fashion as to make connections with other hypothesized words during the decoding phase of automatic speech recognition computationally tractable.

Overview of the method

If we consider a particular word, we find that its phonetic realizations are influenced to some extent by the phonetic character of the words immediately preceding and following it. For example, we may find that a word such as 'those' behaves in different phonetic contexts as represented by the graphs in **Figure 1**. Usually only the first few phones and the last few phones will be affected in this way: The phonetic realizations possible for the central portion of a word longer than a few phones are normally unaffected (or, at most, minimally affected) by the phonetic environment in which the word appears. If a complete set of phonological rules is specified in advance, then it is possible to determine all of the variations that can arise in the phonetic realization of a word as a result of the phonetic character of surrounding words [7]. We would like to distill from a collection of graphs, a single graph, as shown in **Figure 2**. Here, each of the initial nodes has associated with it a *left connection* which is simply a representation of the phonetic environment to which it is applicable. Similarly, each terminal node has associated with it a *right connection* which represents the phonetic environment to which it is applicable.

We describe a method for obtaining these graphs which allows for efficient storage of the resulting graph and for the rapid connection of a number of such graphs to form a graph for a string of words. We illustrate the method with a detailed example. Our attention is focused on the issue of constructing the graphs and not on the phonological rules themselves. We therefore use an abbreviated and simplified set of rules and ignore such things as stress so that our example is not unnecessarily complex. In our example, we use the word 'those' for which the baseform is /|ðoz/, where the vertical bar at the beginning of the baseform is a word boundary. When the word 'those' appears in a string of words, the /z/ at the end of the baseform will be followed immediately by the word boundary at the beginning of the next word, and rules are applied accordingly.

We apply to this baseform the following set of five rules:*

1. ð → d / s| ___
2. ð → dð / f| ___
3. z → ž / ___ |š
4. z → ž / o ___ (|)⟨č, ǰ, j⟩
5. z → s / ___ |⟨f, ð, s, š, č⟩

Each of these rules has a left-hand side (the part before the arrow), a right-hand side (the part between the arrow and the slash), a left-hand context (the part between the slash and the underscore), and a right-hand context (the part after the underscore). In rule 5, the left-hand side is z, the right-hand side is s, the left-hand context is empty, and the right-hand context is |⟨f, ð, s, š, č⟩. The rule says that s is an alternative to z if the z is followed by a word boundary which is in turn followed by any one of f, ð, s, š, or č. Because the left context is empty in this rule, there is no constraint placed on the phones which precede the z. Rule 4 says that ž is an alternative to z when the latter follows o and is itself followed by any one of č, ǰ, or j. The parentheses around the word boundary in rule 4 show that the word boundary is optional. Thus rule 4 will also apply in a situation where the z is at the end of a word, provided the next word begins with any one of č, ǰ, or j. For a detailed discussion of the application of our phonological rules and of the notation we have used in writing them, see [5].

The steps involved in constructing a graph for 'those' are shown in **Figure 3**. We begin with a single node, as shown in Figure 3(a). Since none of our rules applies to the word boundary, we can simply add the word boundary to this node to obtain the graph in Figure 3(b). Either of rules 1 and 2 might apply because the ð in 'those' is immediately preceded by a word boundary. In some contexts, rule 1 will apply but not rule 2, in some contexts rule 2 will apply but not rule 1, and in some contexts neither rule will apply. Specifically, rule 1 will apply exactly when the phone preceding the word boundary is /s/, rule 2 will apply exactly when it is /f/, and neither will apply when the preceding phone is anything else. These considerations lead to the graph shown in Figure 3(c). We have split the initial node into three nodes and labeled each with the left context for which it is appropriate. Again, none of the rules applies to /o/, and so we simply add it, thereby producing the graph

* The phonetic symbols used in the rules are these:

ð as in *then*	z as in *zoo*	č as in *chum*
d as in *bend*	ž as in *vision*	ǰ as in *just*
s as in *sun*	š as in *ship*	j as in *young*
f as in *fox*	o as in *those*	n as in *men*

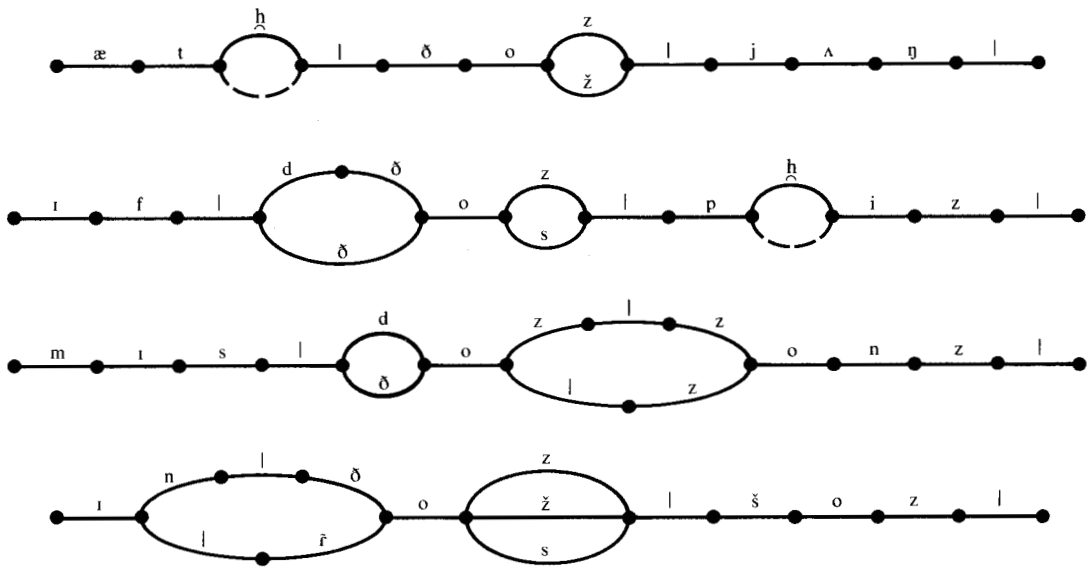

Figure 1

Phonetic graphs for 'those' in different phonetic contexts.

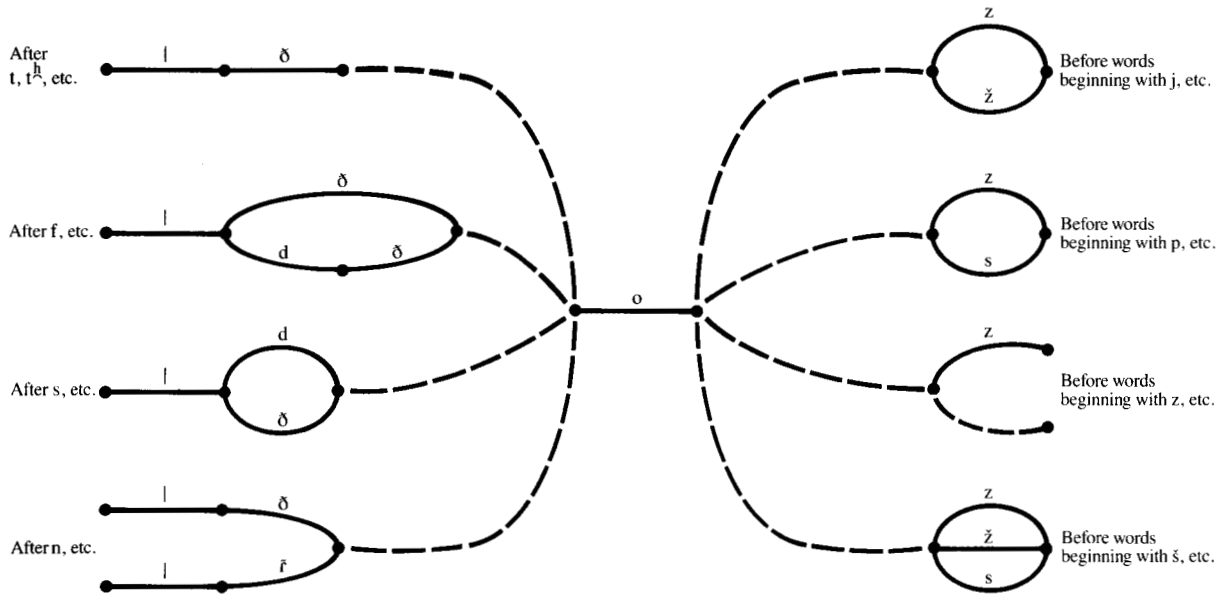

Figure 2

Graph for 'those' allowing different phonetic contexts through multiple initial and final nodes.

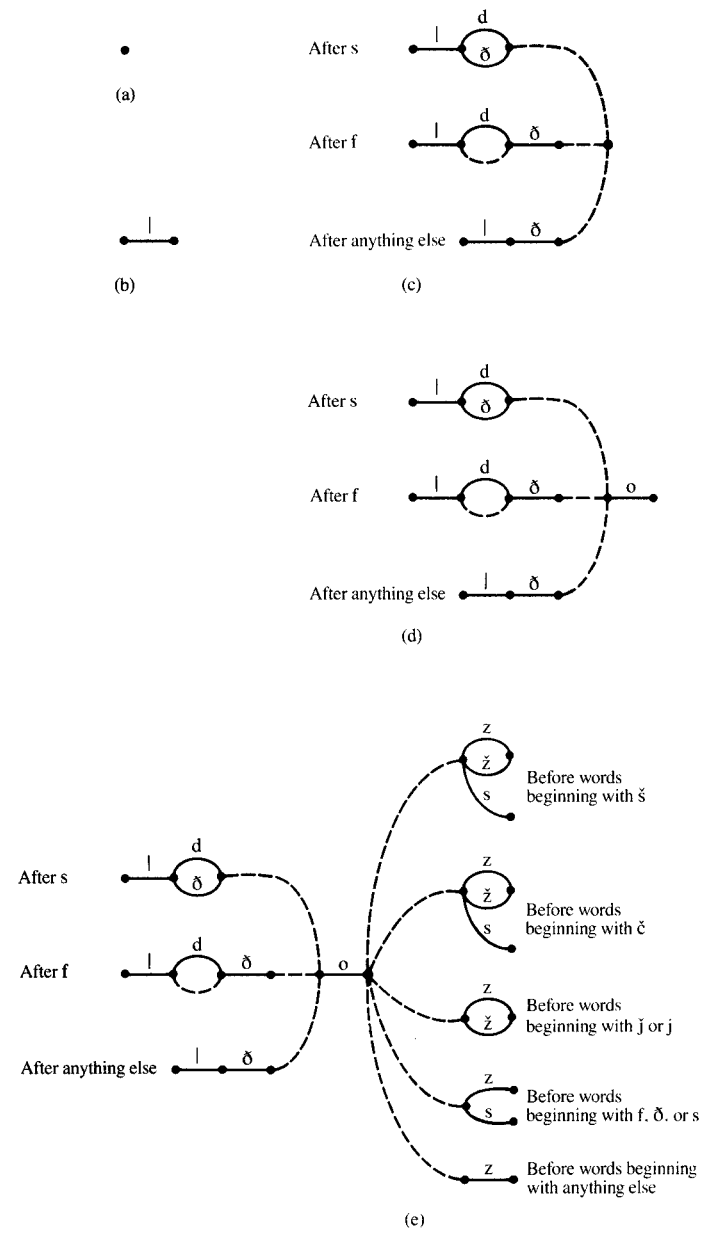

Figure 3

Stages in the generation of a graph for 'those.' (a) Initial node. (b) Graph after processing word boundary. (c–e) Phone-by-phone development of graph.

shown in Figure 3(d). Rules 3, 4, and 5 may each apply to the /z/ in some contexts: Rules 3 and 5 but not rule 4 will apply if the next word begins with /š/; 4 and 5 but not 3 will apply if the next word begins with /č/; only 4 will apply if the next word begins with either /ǰ/ or /j/; only 5 will apply if the next word begins with /f/, /ð/, or /s/; and none of the rules will apply if the next word begins with anything else.

We must therefore make a five-way split in the graph reflecting the possible contexts of the following word. The result is shown in Figure 3(e).

Each of the initial nodes in Figure 3(e) is labeled with a set of conditions which must be satisfied if it is to be used. Each of the final nodes is similarly labeled. If we wish to connect this graph to another to form a graph for a pair of words,

then we need only check, for each possible connection of a final node on one graph to an initial node from another graph, whether all of the necessary conditions are satisfied. For example, suppose we wish to connect a graph for the word 'send' to a graph for the word 'those' to obtain a graph for the word pair 'send those.' A graph for 'send' is shown in **Figure 4**. In obtaining it, we have used the following additional rule:

6. d → ∅ / n __ | ð

This rule states that a final d can be omitted if it follows an n and the next word begins with ð. Actually, this rule is a part of a more general rule (consonant-cluster simplification), but for our purposes this simplified version of it suffices. Now, the top terminal node for 'send' can only be used in front of words that do not begin with ð, and the bottom one can only be used in front of words that do begin with ð. Since 'those' begins with ð, we must use the bottom terminal node from 'send.' It is also clear that we must use the bottom initial node from 'those' because 'send' ends with neither f nor s. Therefore the resulting graph for 'send those' is the one shown in **Figure 5**.

When the graphs are stored as indicated in the examples above, determining which terminal nodes of one graph connect to which initial nodes of another graph is a time-consuming process. We must first determine for which of the terminal nodes of the first graph the baseform for the second graph provides the correct right context. Then we must determine for which of the initial nodes of the second graph this terminal node provides the correct left context. It is this process of selecting which nodes to interconnect and which nodes to prune that we would like to make more efficient. The key to doing so is that there are only a finite number of rules and hence only a finite number of left and right contexts that need to be distinguished. Because of this, we can carry out most of the computation necessary for making a connection ahead of time. Let us return to our example and see how this can be done.

We begin by numbering the word boundaries that appear in each rule as shown below:

1. ð → d / s|^{①} __
2. ð → dð / f|^{②} __
3. z → ž / __ |^{③}š
4. z → ž / o __ (|^{④})⟨č, ǰ, j⟩
5. z → s / __ |^{⑤}⟨f, ð, s, š, č⟩
6. d → ∅ / n __ |^{⑥}ð

Although each of our rules has exactly one word boundary, in general a rule may have any number of word boundaries or none at all. (It is therefore an artifact of our example that the numbers on the word boundaries correspond to the numbers of the rules themselves.) For word boundaries which appear in the left-hand context of a rule, we let the number of the word boundary represent the statement that *the graph to the left of a position satisfies the portion of the left-hand context to the left of that word boundary*; and we let the negative of the number represent the statement that this is not the case. Thus, instead of putting s as a condition on the first initial node of 'those,' we label the node with the pair (−2,1), indicating that this node can only be connected to a terminal node which provides the context required by word boundary 1 and does not provide the context required by word boundary 2. We call this label the *left requirement* of the node. Similarly, for word boundaries which appear in the right-hand context of a rule, we let the number of the word boundary represent the statement that *the graph to the right of a position satisfies the portion of the right-hand context to the right of the word boundary*; and we let the negative of the number represent the statement that this is not the case. The conditions necessary for attaching a particular terminal node can then be expressed as a set of positive and negative numbers which we call the *right*

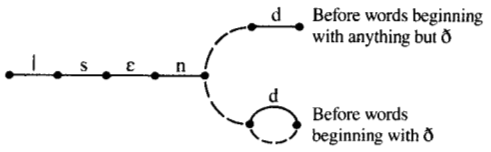

Figure 4

Graph for 'send.'

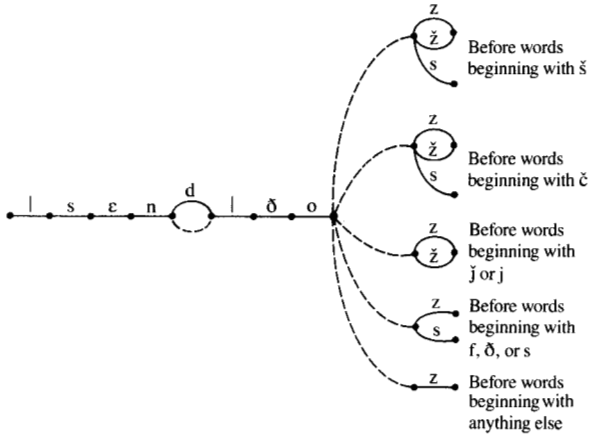

Figure 5

Graph for 'send those.'

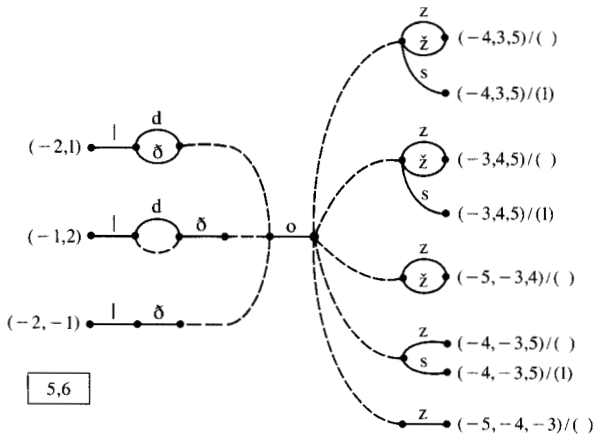

Figure 6

Graph for 'those' with left and right requirements and context descriptions.

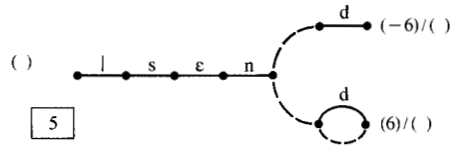

Figure 7

Graph for 'send' with left and right requirements and context descriptions.

requirement of the node. The top terminal node of 'send,' for example, has the right requirement (−6) because it can only be used when the following word does not provide the context following word boundary 6.

In addition to providing each node with a left or right requirement, we must also indicate the set of requirements which it supplies. Because our rules are constructed so that the right context is taken from the as-yet-unprocessed baseform, all of the initial nodes supply the same requirements, which we call the *left context description* of the baseform. Each terminal node, however, has its own *right context description* because different terminal nodes may supply differing requirements depending on the details of the graph to the left. In **Figures 6** and **7**, the left requirement of each initial node is shown in parentheses next to it; the left context description is shown in a box below the graph; and the right requirement and right context description of each terminal node are separated by a slash and placed to the right of the node. The left context description of 'those' is (5,6). This means that a terminal node from a graph can be connected to 'those' only if its right requirement includes no positive numbers other than 5 or 6 and includes neither −5 nor −6. The top terminal node of 'send' includes −6 and therefore is not eligible for connection, but the bottom terminal node includes only 6 and therefore is eligible for connection. The right context description of the bottom terminal node of 'send' is empty because this node satisfies none of the word boundaries that appear in left-hand contexts. Therefore, the only initial node from 'those' which we can connect to it is the bottom one for which the left requirement is (−2,−1).

The example above suggests a general procedure for connecting the graphs for two baseforms. We say that a requirement and a context description are compatible if each of the positive numbers in the requirement appears in the context description and if, further, the negative of no number in the context description appears in the requirement. Given two words, we say that a terminal node of the first word is eligible for connection to the second word if its right requirement is compatible with the left context description of the second word. We say that an initial node of the second word is eligible for connection to a particular terminal node of the first word if the left requirement of the initial node is compatible with the right context description of the terminal node. To connect two words, we find all of the terminal nodes of the first word which are eligible for connection and then we connect to each of these eligible terminal nodes the initial node of the second word which is eligible for connection to it. It is easy to show that there is exactly one eligible initial node for any eligible terminal node.

So far, none of the rules that we have dealt with has had a word boundary on either the left- or right-hand side. Many important phonological modifications are best accounted for by rules that change things on both sides of a word boundary. To show how we handle this kind of rule, we extend our example by adding one more rule:

7. n d $|^{⑦}$ð → $|^{⑧}$ɾ̃ / __

This rule says that a single flapped n is an alternative to the entire sequence on the left-hand side anywhere the latter appears. It can be applied at the beginning of 'those' and also at the end of 'send.' We expect, therefore, that its presence will modify the initial nodes of 'those' and the terminal nodes of 'send.' The word boundary on the left-hand side will be number 7, and the word boundary on the right-hand side number 8. The graphs for 'those' and 'send' when we use this additional rule are shown in **Figures 8** and **9**. We see that the graph for 'those' has all of the initial nodes that it

had before and that the left requirement for each of them has been augmented with (−7). In addition, there are two new initial nodes. The first has as its left requirement (−8,−2,−1,7). This node corresponds to the case where our new rule is applicable but the alternative presented by the right-hand side has not been chosen. The second has as its left requirement (−2,−1,7,8) and corresponds to the case where the new rule is applicable and the alternative has been chosen. The left context description for 'those' has also been augmented with (7).

In the graph for 'send,' we see that all of the original terminal nodes are still present and that their right requirements and right context descriptions are unmodified. There are, however, two new terminal nodes corresponding to the two new initial nodes of 'those.' Notice also that all of the terminal branches of 'send' begin earlier because now the n is subject to change depending on the following context.

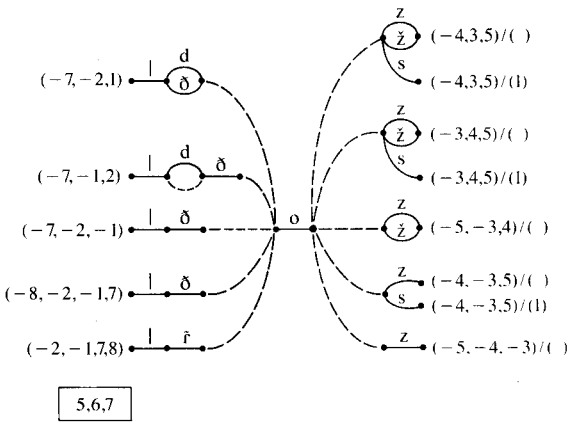

Figure 8

Graph for 'those' including the effect of rule 7.

• *Efficient storage*

We can store graphs such as those in Figures 8 and 9 as a collection of nodes and transitions. Each of the requirements and context descriptions can be stored as a list of numbers with an indication of the node for which it is appropriate. While such a scheme is straightforward, it is possible to achieve a much more efficient representation of these graphs. In order to describe our scheme, we need some terminology.

We refer to nodes through which each path must pass as *confluent nodes*. For example, in Figure 8 the nodes immediately preceding and following o are confluent nodes. The portion of the graph between consecutive confluent nodes is called a *confluent link*, or *c-link*. The leftmost confluent node is called the first confluent node and the rightmost one is called the last confluent node. The part of the graph to the left of the first confluent node is called the *left hook*, and the part to the right of the last confluent node is called the *right hook*. The left hook includes the left context description and also the left requirements of the initial nodes, while the right hook includes the right context descriptions and the right requirements of the terminal nodes. We can then store a graph simply as a left hook, a sequence of c-links, and a right hook. Even for a large vocabulary, the number of different left and right hooks is less than 100, and the number of different c-links is only on the order of 1000. If we add a new word to the vocabulary, the chance that we will require a new hook or c-link is small. Therefore, as the vocabulary grows, the marginal cost of storing one additional word becomes quite small, being roughly equal to the cost of storing the spelling for the word. A quantitative discussion of the storage requirements for a particular 1000-word vocabulary is given later.

• *Rapid application*

During decoding, it is necessary to be able to construct the graph for a sequence of words rapidly. If we store graphs as

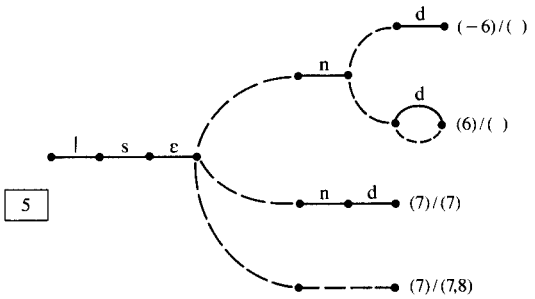

Figure 9

Graph for 'send' including the effect of rule 7.

indicated above, we will have no trouble with the portion of the graph between the first and last confluent nodes of a word. The challenge comes in constructing the portion of the graph from the last confluent node of one word to the first confluent node of the following word. This part of the graph is uniquely determined if we know the right hook of the first word and the left hook of the second word. Since any word can follow any other word, we must be prepared to connect any right hook to any left hook. We call the graph that corresponds to a right-hook/left-hook pair the *boundary link*, or *b-link*, for the pair. Because the number of different hooks is small, we can compute all of the necessary b-links before

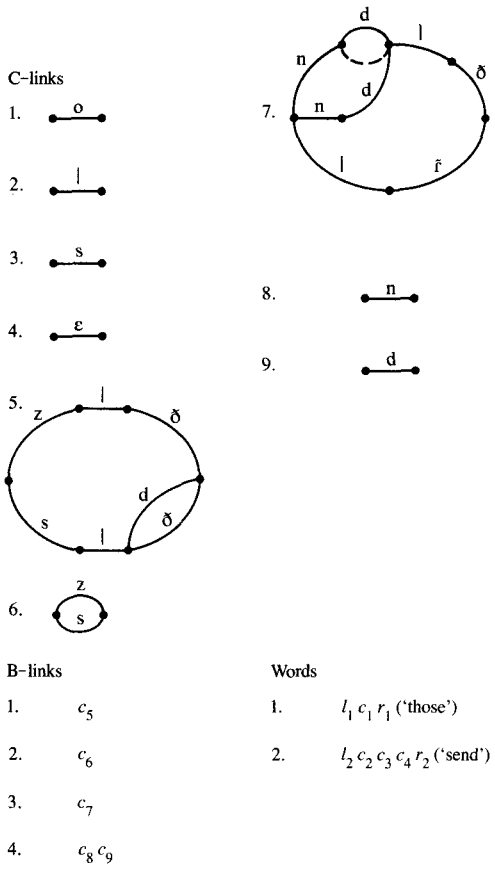

Figure 10

Complete information for the vocabulary comprising 'send' and 'those.'

decoding. Each b-link is a graph with a sequence of confluent nodes and can be stored as a series of c-links. If there are n_l left hooks and n_r right hooks, then there will be a total of $n_l n_r$ b-links. If we assign the index $(l-1)n_r + r$ to the b-link corresponding to left hook l and right hook r, then the b-links will have indices ranging from 1 to $n_l n_r$.

An example will make the process of connection clear. Suppose that the vocabulary has only the two words 'send' and 'those.' Then when we are decoding, we will need only the data shown in **Figure 10**. Notice that we store nothing to indicate the structure of the hooks, because by storing the b-links explicitly we eliminate the need for this. We wish to construct a graph for word 2 followed by word 1 ('send those'). We look up words 1 and 2 and find that word 2 has right hook 2 (r_2 in the figure) and word 1 has left hook 1. Therefore, since $n_r = 2$, we need b-link 2 to fill the space between the last confluent node of 'send' and the first confluent node of 'those.' The resulting graph has left hook l_2 followed by the c-link sequence $c_2 c_3 c_4 c_6 c_1$ followed by right hook r_1.

• *A numerical example*

In order to illustrate the efficiency of storage possible with the method described here, we have made a comparison between our technique and a straightforward method of storage for a vocabulary of 1200 words. These are the most frequent words in a large collection of text from the descriptive portions of U.S. patents in laser technology. As such they are not entirely representative of what one might call everyday English (for example, the average word is almost one letter longer than the average over all of English), but we believe that they are sufficiently similar to allow a valid comparison. For each word, we have constructed a graph using the phonological rules described in [5].

Our 1200 words have an inventory of 1065 c-links, 82 right hooks, 62 left hooks, and 5084 b-links. We assume that a graph can be stored with 4 bytes for each node and 4 bytes for each arc. If we store the graph for each word directly, then, under this assumption, we will require a total of 262244 bytes. If, on the other hand, we store graphs only for the c-links but keep words and b-links as sequences of c-links, then these c-link graphs will need only 114780 bytes. In addition, we will need 17374 bytes to store the left and right hooks for each word and the list of c-links that makes up its central portion, and 26442 bytes to store the c-link sequences for the b-links. Altogether, then, we will need 158596 bytes using our method as compared with 262244 bytes using the straightforward method, or a saving of about 40%.

A saving of 40% on 1200 words, however, does not really indicate the storage efficiency which our method allows. When we add a word to a large vocabulary, we may already have many of the c-links that we require. Only rarely will a new word involve a new left or right hook. Thus we can expect that eventually the storage required for new c-links and b-links will grow very slowly and that when we add a new word we will need only the storage required for it. Thus, in addition to comparing the total storage, we should look at the marginal growth rates in storage for the two methods.

In **Figure 11**, we show, as a function of the vocabulary size, the average number of new c-links and b-links required when a new word is added to the vocabulary. We have estimated these growth rates by choosing a random sample from our 1200 words; determining the number of c-links and b-links required to represent it; choosing an additional random sample of 10 words from those not chosen in the first sample; and, finally, determining for these 10 words the number of additional c-links and b-links which they require. Each of the plotted values is the average of 1000 such trials. We see that when the vocabulary reaches 1000 words, each new word requires only about 0.4 new c-links and 2.6 new b-links.

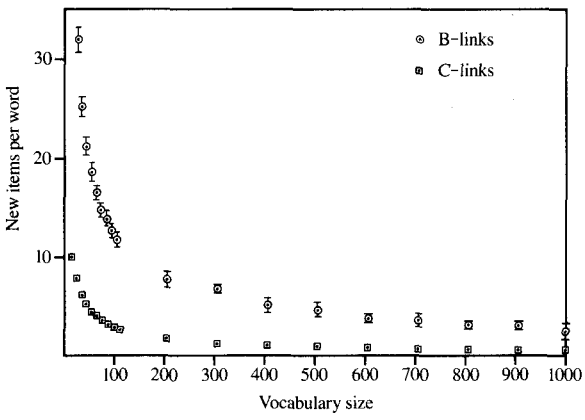

Figure 11

Number of new items per word vs. vocabulary size.

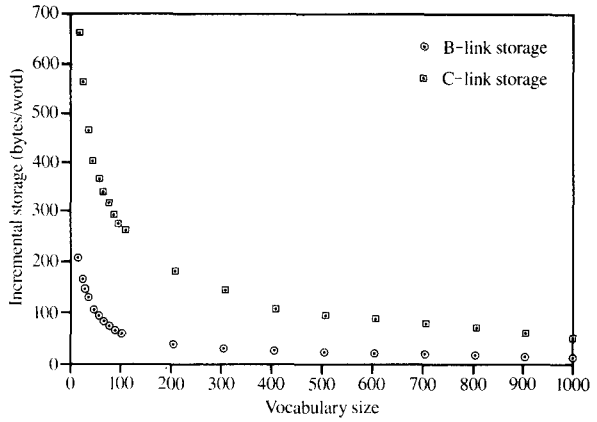

Figure 12

Storage growth rate vs. vocabulary size.

In **Figure 12**, we show similar results for growth rates in storage. As we have pointed out before, left and right hooks require no storage, and so the graph shows only the growth rate in storage for c-links and b-links. After 1000 words, each new word requires about 53 bytes of c-link storage and 13 bytes of b-link storage. Since the word itself requires about 14 bytes of storage, independent of vocabulary size, we need altogether about 80 bytes as compared with about 220 bytes for the straightforward method. Thus, after 1000 words the marginal storage required is reduced by 64%. We expect that for very large vocabularies the growth rates for c-link and b-link storage will continue to fall and the marginal storage required for a new word will approach 14 bytes, or about 94% less than the straightforward method.

References and note

1. John G. Kemeny and J. Laurie Snell, *Finite Markov Chains*, D. Van Nostrand Company, Inc., Princeton, NJ, 1960.
2. N. R. Dixon and C. C. Tappert, "Some Problems in the Derivation of a Phonetic Referent for Evaluation and Lexical Representation in Automatic Recognition of Continuous Speech," paper presented at the 84th meeting of the Acoustical Society of America, Miami, FL, November 1972. (Abstract available in program of meeting.)
3. C. C. Tappert and N. R. Dixon, "Application of Sequential Decoding for Establishing Lexical Variants by Rule in Automatic Recognition of Continuous Speech," paper presented at the 84th meeting of the Acoustical Society of America, Miami, FL, November 1972. (Abstract available in program of meeting.)
4. N. R. Dixon and C. C. Tappert, "Intermediate Performance Evaluation of a Multi-Stage System for Automatic Recognition of Continuous Speech," *Technical Report RADC-TR-72* (prepared by IBM), Rome Air Development Center, Griffiss Air Force Base, NY, 1973.
5. Paul S. Cohen and Robert L. Mercer, "The Phonological Component of an Automatic Speech-Recognition System," *Speech Recognition*, D. R. Reddy, Ed., Academic Press, Inc., New York, 1975, pp. 275–320.
6. F. Jelinek, R. L. Mercer, and L. R. Bahl, "Continuous Speech Recognition: Statistical Methods," *The Handbook of Statistics*, Vol. 2, P. L. Krishnia and L. N. Kanal, Eds., North-Holland Publishing Company, Amsterdam, 1982, pp. 549–573.
7. Obviously, in practice, no set of phonological rules for a language is truly complete—partly because linguistic knowledge is itself incomplete and, more importantly, because infinite phonetic detail would theoretically be required.

Received November 5, 1985; accepted for publication September 30, 1986

Robert L. Mercer *IBM Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Mercer received the B.S. degree in physics and mathematics from the University of New Mexico, Albuquerque, in 1968, and the M.S. and Ph.D. degrees in computer science from the University of Illinois, Urbana, in 1970 and 1972, respectively. Since 1972 he has been a Research Staff Member, Computer Sciences Department, and is currently manager of Real-Time Speech Recognition at the Thomas J. Watson Research Center. Dr. Mercer is a member of Phi Beta Kappa, Phi Kappa Phi, and Sigma Xi.

Paul S. Cohen *IBM Journal of Research and Development, 500 Columbus Avenue, Thornwood, New York 10594.* Mr. Cohen received the B.A. in linguistics and comparative literature from the City College of New York in 1964, and the M.A. in linguistics (with a specialization in English) from Columbia University in 1970. From 1963 to 1965, he was an editor on *The Random House Dictionary of the English Language*, and from 1965 to 1968 did linguistic research on a U.S. Office of Education grant at Columbia University. Mr. Cohen joined the Research Division of IBM at Yorktown Heights in 1968 as an Instructional Programmer, and worked chiefly in

computer-assisted instruction and linguistics. From 1970 to 1972, he did linguistic research and teaching-material development under a Ford Foundation Grant at Brooklyn College of the City University of New York. He then rejoined IBM at Yorktown, where, as a Research Staff Member, he did research in the areas of speech recognition and speech synthesis until 1983. He worked as a computational linguist for the E/ME/A National Language Support Center and was a Program Manager for World Trade International Technical Support in Poughkeepsie from 1984 to 1985. At present, he is an Associate Editor of this journal. At IBM, Mr. Cohen has received a Research Division Award for contributions to speech recognition, an Invention Achievement Award, and a First Patent Application Award.