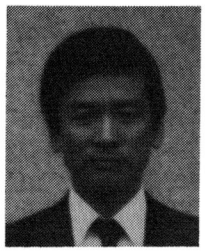
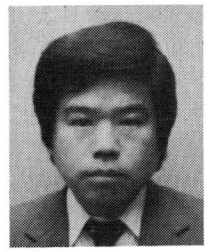
[14] R. Taniguchi *et al.*, "Picture understanding and retrieving system of weather chart," in *Proc. 6th ICPR*, 1982.
[15] E. Kawaguchi *et al.*, "An information understanding system of basic weather report," *Trans. IECE Japan*, vol. E64, pp. 71-78, Feb. 1981.



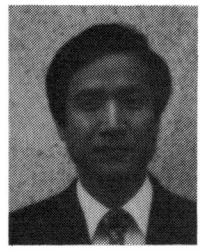
Eiji Kawaguchi was born in Japan on November 27, 1940. He received the B.E., M.E., and D.E. degrees in 1964, 1966, and 1971, respectively, from Kyushu University, Fukuoka, Japan.

Currently he is working as an Associate Professor at the Interdisciplinary Graduate School of Engineering Sciences, Kyushu University. His main technical interest pertains to speech recognition systems, digital picture processing, computer systems, etc.



Tsutomu Endo was born in Japan on April 30, 1949. He received the B.E., M.E., and D.E. degrees from Kyushu University, Fukuoka, Japan, in 1972, 1974, and 1979, respectively.

He is now working as an Associate Professor at the Faculty of Engineering, Oita University, Oita, Japan. His current research interests include natural language processing and digital picture processing.



Jun-ichi Matsunaga was born in Nagasaki, Japan, on November 20, 1946. He received the B.E. degree from Kyushu University, Fukuoka, Japan, in 1972.

He started to work for the Department of Computer Science and Communication Engineering, Kyushu University, in 1974. He is presently a Research Associate at the Department of Information Systems, Interdisciplinary Graduate School of Engineering Sciences, Kyushu University. His current research interest is in speech recognition, speech synthesis, digital signal processing, and computer systems.

An Integrated Algorithm for Text Recognition: Comparison with a Cascaded Algorithm

JONATHAN J. HULL, SARGUR N. SRIHARI, MEMBER, IEEE, AND RAMESH CHOUDHARI

Abstract—**The use of diverse knowledge sources in text recognition and in correction of letter substitution errors in words of text is considered. Three knowledge sources are defined: *channel characteristics* as probabilities that observed letters are corruptions of other letters, *bottom-up context* as letter conditional probabilities (when the previous letters of the word are known), and *top-down context* as a lexicon. Two algorithms, one based on integrating the knowledge sources in a single step and the other based on sequentially cascading bottom-up and top-down processes, are compared in terms of computational/storage requirements and results of experimentation.**

Index Terms—**Contextual pattern recognition, dictionary organization, error correction, text recognition, trie, Viterbi detection.**

I. Introduction

THE expanding role of computers in the manipulation of text has focused attention on automatic detection and correction of errors. Textual errors are predominantly caused during the input process, e.g., errors are committed by an optical character reader (OCR) in interpreting text from hard copy [2], [16], [24], typographical errors are caused during keyboard entry [23] and spelling errors due to human ignorance are caused during text creation [13], [15].

Methods of correcting textual errors use some form of either local or global contextual knowledge and can be characterized as *data-driven* or *bottom-up*, *concept-driven* or *top-down*, and *mixed* or *hybrid* [6]. Data-driven algorithms for text recognition/error correction refine successive hypotheses about an input string. An example is a program that uses a statistical (Markovian) representation of contextual knowledge as a table of *transition probabilities*, i.e., the probability of each letter given that a set of letters have occurred previously. These algorithms typically rely on the assumptions that correct text can be adequately described by such a table and that the process by which errors are introduced can be represented by a statistical model.

Concept-driven algorithms begin with an expectation of what the input string is likely to be and fit the data to this expectation. Examples are algorithms that use implicit or explicit representations of dictionaries, syntax, and semantics.

Manuscript received December 8, 1981; revised November 4, 1982. This work was supported by the National Science Foundation under Grant IST-80-10830.

J. J. Hull and S. N. Srihari are with the Department of Computer Science, State University of New York at Buffalo, Amherst, NY 14226.

R. Choudhari was with the Department of Computer Science, State University of New York at Buffalo, Amherst, NY 14226. He is now with the Department of Computer Science, Claflin College, Orangeburg, SC 29115.

0162-8828/83/0700-0384$01.00

The performance of algorithms based on a pure bottom-up or top-down approach usually falls short of human performance. As a consequence, several *hybrid* methods that attempt to use both bottom-up and top-down contextual knowledge to achieve human expertise have been proposed.

Hybrid approaches for text recognition/error correction may merge a bottom-up refinement process based on the utilization of transition probabilities together with a top-down process based on searching a dictionary. Hybrid algorithms can be classified as *integrated* and *cascaded*. The former is characterized by an integration of bottom-up and top-down knowledge sources while the latter uses a layered method that effects sequential rather than simultaneous utilization of information. An example of a cascaded algorithm is one that processes textual data by performing classification based on shape features and letter transition probabilities and then postprocesses the resultant to fit a dictionary entry. The issue of integrated versus cascaded in text recognition is analogous to a similar issue in language understanding—a cascaded approach separates syntactic and semantic processes while an integrated approach merges these processes [3], [17].

The objective of this paper is to describe a hybrid-integrated algorithm, an earlier version of which was given in [20], and to evaluate it with respect to the hybrid-cascaded algorithm described in [18]. This evaluation is in terms of theoretical as well as experimental characteristics. Each algorithm described is applicable to recognizing multifont text as well as correcting textual errors where one character is substituted for another; the latter phenomenon occurs in any text or speech input system where it can be assumed that every input word is properly segmented, e.g., when machine-printed text is read by an OCR.

II. Problem Formulation

Let the observed word be $\boldsymbol{X} = X_0X_1 \cdots X_mX_{m+1}$ where the letters $X_i (1 \leqslant i \leqslant m)$ belong to an alphabet $\boldsymbol{L} = L_1, \cdots, L_r$, and X_0 and X_{m+1} are special letters known as delimiters (blank, hyphen, etc.) which are used to separate different words of text. The probability that a word $\boldsymbol{Z} = Z_0Z_1 \cdots Z_mZ_{m+1}$ could have caused $\boldsymbol{X}$ is given from Bayes' decision theory as the *a posteriori* probability

$$P(\boldsymbol{Z}|\boldsymbol{X}) = [P(\boldsymbol{X}|\boldsymbol{Z}) * P(\boldsymbol{Z})]/P(\boldsymbol{X})$$

where $P(\boldsymbol{X}|\boldsymbol{Z})$ is the probability of observing $\boldsymbol{X}$ when $\boldsymbol{Z}$ is the true word, $P(\boldsymbol{Z})$ is the *a priori* probability of $\boldsymbol{Z}$ and $P(\boldsymbol{X})$ is the probability of string $\boldsymbol{X}$. Since $P(\boldsymbol{X})$ is independent of $\boldsymbol{Z}$, the word $\boldsymbol{Z}$ that maximizes $P(\boldsymbol{Z}|\boldsymbol{X})$ can be determined by maximizing the expression

$$G(\boldsymbol{X}|\boldsymbol{Z}) = \log P(\boldsymbol{X}|\boldsymbol{Z}) + \log P(\boldsymbol{Z}). \tag{1}$$

The objective is to determine that word $\boldsymbol{Z}$ in a set of words D that maximizes (1); this problem is referred to as one of *text error correction*. The set D is determined by contextual constraints and is known as a *dictionary*. The term dictionary is used to refer to a list of words where the words are not associated with descriptive information such as meanings, derivations, etc.

The above problem statement assumed that X_i are hard decisions output by a classifier. In a more general formulation, called the *text recognition* problem, $\boldsymbol{X} = \boldsymbol{x}_0\boldsymbol{x}_1 \cdots \boldsymbol{x}_m\boldsymbol{x}_{m+1}$ where $\boldsymbol{x}_i$ is a feature vector representing letter X_i. Whether $\boldsymbol{X}$ is a string of letters or a string of feature vectors, the computation is one of maximizing (1). The only difference is that in storing the $P(\boldsymbol{X}|\boldsymbol{Z})$ distribution in memory, there is a much larger set of possibilities for feature vectors than for letters; even in the latter case the number of possible m - letter strings is r^m. Thus storing the $P(\boldsymbol{X}|\boldsymbol{Z})$ distribution in memory is impractical because of the large number of possibilities for $\boldsymbol{X}$ and $\boldsymbol{Z}$. If conditional independence is assumed to exist among $X_0, X_1, \cdots, X_{m+1}$, then

$$\log P(\boldsymbol{X}|\boldsymbol{Z}) = \sum_{i=0}^{m+1} \log P(X_i|Z_i) \tag{2}$$

and the storage for the $P(\boldsymbol{X}|\boldsymbol{Z})$ distribution is reduced to the order of r^2.

According to this assumption the observed letters are independent of each other, which is valid for printed text but not necessarily for cursive script. The probability $P(X_i|Z_i)$, called the *confusion probability*, is the probability of observing letter X_i when the true letter is Z_i.

Bottom-Up Methods

A bottom-up approach to text recognition may assume that the English language is an nth order Markov source, i.e., that the identity of a character can be predicted given that the identity of the previous n characters is known. If it is assumed that words are generated by an nth order Markov source, then the *a priori* probability $P(\boldsymbol{Z})$ can be expressed as

$$P(\boldsymbol{Z}) = P(Z_{m+1}|Z_{m+1-n} \cdots Z_m) \cdots P(Z_1|Z_0) * P(Z_0) \tag{3}$$

or

$$\log P(\boldsymbol{Z}) = \log P(Z_{m+1}|Z_{m+1-n} \cdots Z_m) + \cdots + \log P(Z_1|Z_0) + \log P(Z_0) \tag{4}$$

where $P(Z_k|Z_{k-n} \cdots Z_{k-1})$ is called the *nth order transition probability*, i.e., the probability of observing Z_k when the previous n letters are $Z_{k-n} \cdots Z_{k-1}$.

For $n = 1$, $P(\boldsymbol{Z}) = P(Z_{m+1}|Z_m) \cdots P(Z_1|Z_0) * P(Z_0)$, and the word $\boldsymbol{Z}$ with maximum *a posteriori* probability is one that maximizes

$$G_1(\boldsymbol{X}, \boldsymbol{Z}) = \sum_{i=1}^{m+1} \log P(X_i|Z_i) + \log P(Z_i|Z_{i-1})$$

where it is assumed that $P(X_0|Z_0) = P(X_{m+1}|Z_{m+1}) = 1$, i.e., the delimiter symbol is perfectly recognized. For $n = 2$, the corresponding expression is

$$G_2(\boldsymbol{X}, \boldsymbol{Z}) = \sum_{i=1}^{m+1} \log P(X_i|Z_i) + \log P(Z_i|Z_{i-2}Z_{i-1})$$

where $P(Z_1|Z_{-1}Z_0) = P(Z_1|Z_0)$.

The *Viterbi algorithm* (VA) [7], [14], as it is used for text

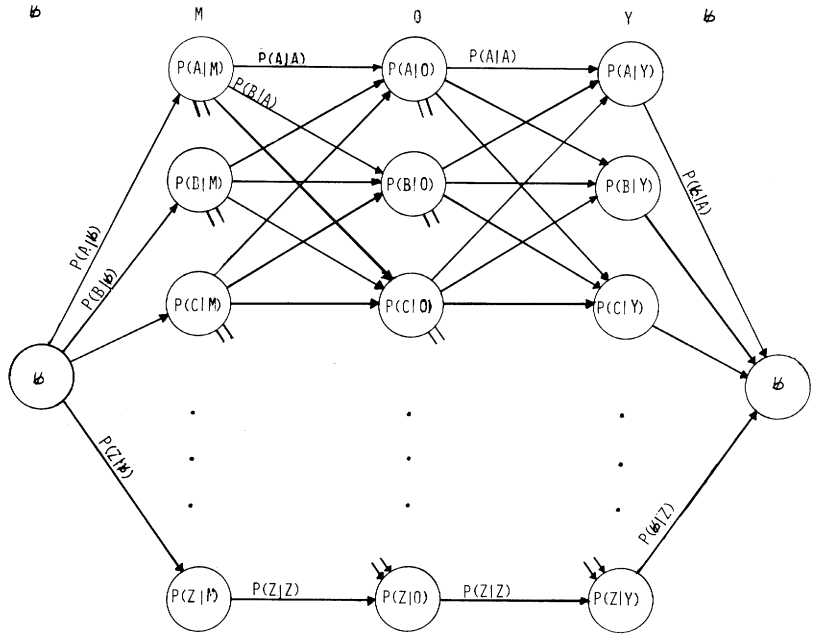


Fig. 1. An example of path tracing by the Viterbi algorithm.

recognition, finds the word $\boldsymbol{Z}$ that maximizes $G_i(\boldsymbol{X}, \boldsymbol{Z})$ over all possible m-letter strings $\boldsymbol{Z}$, not necessarily those in a dictionary. The method is based on a dynamic programming formulation that leads to a recursive algorithm.

If L_j represents the jth letter of the alphabet, then max $[G_i(X_1 \cdots X_k, Z_1 \cdots Z_{k-1}Z_k = L_j)]$ over all possible values of $Z_1 \cdots Z_{k-1}$ can be computed trivially if the r values corresponding to max $[G_i(X_1 \cdots X_{k-1}, Z_1 \cdots Z_{k-2}Z_{k-1} = L_p)$, $p = 1, \cdots, r$ are known for all possible values of $Z_1 \cdots Z_{k-2}$. This formulation reduces the complexity to $O(r^2)$ and $O(m)$ from the $O(r^m)$ required by an exhaustive search. The algorithm can be viewed as a shortest path algorithm through a directed graph of $r \times m$ nodes called a trellis. The log-transition probabilities are associated with the edges of the trellis and the log-confusion probabilities are associated with the nodes. The cost of a path is then the sum of all the edge and node values in the path.

The VA as it is used for text recognition is illustrated in Fig. 1 where the correct word is *MAY* and *O* has been substituted for *A* to produce *MOY* as input. The cost of the correct path is: $\log P(M|\not b) + \log P(M|M) + \log P(A|M) + \log P(O|A) + \log P(Y|A) + \log P(Y|Y) + \log P(\not b|Y)$. If this sum was the highest for all the paths in the trellis, *MAY* would be output; otherwise another set of letters would be chosen.

Some heuristic modifications of the VA that decrease its complexity but attempt to retain its recognition capability have used either a fixed number of alternatives less than r, called the modified Viterbi algorithm (MVA) [19] or a variable number of alternatives [5] for each Z_k. These alternatives can be determined by the letters that have the highest confusion probability.

The VA (and its variations) is a purely bottom-up approach whose performance may be unacceptable because the resulting strings do not necessarily belong to a dictionary. For example, in experimentation with the MVA [8], the best word correction rate was 46 percent using second-order statistics (transition probabilities) and 20 percent with first-order statistics, i.e., 46 and 20 percent of the garbled words were transformed into correct dictionary words, while the remainder were either transformed into other garbled words or incorrect dictionary words.

Top-Down Methods

Top-down approaches to using contextual information in text recognition may use some representation of a dictionary. For each input word $\boldsymbol{X}$ they typically maximize a *score* [2] or minimize a *distance measure* [25] over all, or a subset of, the words in the dictionary. The top-down method due to Bledsoe and Browning [2], also called a dictionary look-up method, is as follows. Given m feature vectors $\boldsymbol{x}_i$ $(1 \leqslant i \leqslant m)$ the method computes the score for each dictionary word of length $m, \boldsymbol{Z} = Z_1, \cdots, Z_m$ as

$$\text{score } (\boldsymbol{Z}) = g(Z_1, \boldsymbol{x}_1) + \cdots + g(Z_m, \boldsymbol{x}_m) \tag{5}$$

where $g(L_j, \boldsymbol{x}_i)$ is the confidence, in some sense, that given the feature vector $\boldsymbol{x}_i$, L_j is the true letter. The output word is one that maximizes the score over all $\boldsymbol{Z}$.

Top-down methods, which have high recognition/correction rates, are useful when the vocabulary is limited. However, their use may be impractical due to a linear increase in time complexity with the number of words in the dictionary [21], and even a quadratic increase with the number of characters in each word [25]. Careful choice of a data structure for the dictionary, e.g., a graph structure such as a letter trie [9], may allow the complexity of a top-down method to be limited.

Hybrid Methods

Methods proposed to overcome the drawbacks of the bottom-up or top-down algorithms combine the two approaches in such a way that the constraints of the bottom-up process

are invoked while the error correction capability of the top-down procedure is achieved or exceeded. These algorithms can be said to more closely model the cognitive behavior of a human text interpreter [17].

The hybrid approach suggested in [18] and [21] is a combination of the MVA and the dictionary look-up method of Bledsoe and Browning. It is referred to as the predictor-corrector algorithm (PCA). In the PCA the dictionary is partitioned into blocks of words of equal length and words are sorted according to a *value* determined by (4). Given $\boldsymbol{X}$ as input, the MVA *predicts* an output sequence of length m, $\boldsymbol{Z}$. Then the value of $\boldsymbol{Z}$ is computed by (4) and a binary search determines whether this value is associated with an m-letter dictionary word. If it is, $\boldsymbol{Z}$ is output, otherwise the scores of a subset of the m-letter dictionary words are computed using (5) and the one with the maximum score is output. Thus the output of the MVA is input to a dictionary method and the bottom-up and top-down algorithms are cascaded.

The hybrid-integrated algorithm discussed in the next section recognizes words of text on a character by character basis using bottom-up information about the identities of neighboring characters and top-down information about allowable dictionary words. It uses a graph searching approach to text recognition and a graph-like representation of a dictionary to achieve a natural integration of bottom-up and top-down processing.

III. An Integrated Algorithm

The dictionary Viterbi algorithm (DVA) [20], uses the basic Viterbi algorithm for text recognition combined with a dictionary represented as a letter-trie. The integration of both knowledge sources is attained by the simultaneous search of the VA trellis and the trie. This allows the optimality of the VA to be retained, i.e., producing the maximum *a posteriori* probability string based on a Markov assumption, while constraining the search to a set of words.

The Trie

The data structure used to represent the dictionary and the method used to access words in it is critical to the efficiency of any text recognition algorithm. Several alternative structures are possible [1], [15], [23]; the choice has to be based on the search strategy of the algorithm and the memory available.

A data structure designed for determining whether a given string is a prefix of a dictionary word and thus suitable for use with the VA, is known as the *trie*. The trie and its variations are discussed at length by Knuth [11] and a text error correction system that uses this data structure has been described by Muth and Tharp [13].

The trie considers words as ordered lists of characters, elements of which are represented as nodes in a binary tree. Each node has five fields: a token (character of a dictionary word), CHAR; an array of bits that indicates word length, WL; an end of word tag, E; and two pointers labeled NEXT and ALTERNATE (see Fig. 2).

A node with token L_j is a NEXT descendant (son) of a node with token L_i if L_iL_j is a substring of a dictionary word. A node with token L_j is an ALTERNATE descendant of a node with token L_i if L_j is an alternative for L_i in a dictionary word, where the letters preceding L_j and L_i are specified by the most immediate ancestor node that is a next descendant. Further, it is required that if a node with token L_j is an ALTERNATE descendant of a node with token L_i, then L_j is ordinally greater than L_i; that is, L_j appears later than L_i in an ordering of the elements of the alphabet. The mth bit of WL is set when the token of its node is part of an m-letter word in the trie. The end of word bit is set when the token of its node and the initial substring given to reach the node are a complete dictionary word.

Accessing the trie to determine the legality of a word prefix is efficient compared to the extraction of the same information from a dictionary organization such as a sequential structure or a hash table. A Boolean function for this task, called ACCESS_TRIE (see [20]) has the pointer to a trie node (representing the initial substring of a dictionary word), the character of a word and its length as input. The function returns **false** when the given pointer has no object, the lexical value of the token in the node pointed to is greater than that of the input character, or the node-token equals that input and the node is not part of a word with the same length as the input. The function returns **true** when the above conditions are not satisfied and the input character is the same as the token in the node pointed to. As a side effect, in this case the pointer that was input is returned as the pointer to the NEXT descendant of this node. If neither the true nor the false conditions are satisfied, a recursive call is made with the same character and length but with the pointer to the ALTERNATE descendant of this node as input.

To determine if a word is in the dictionary, ACCESS_TRIE is called in left to right order for each of the characters in the word. These characters, the length of the word, and a pointer that is updated as described above, are input to each call. The pointer initially points to the root of the trie. The result of the last call determines whether the word is represented in the trie. The time requirements of ACCESS_TRIE are discussed in Section V and experimental observations of their effect on the algorithm are given in Section VI.

The Algorithm

Simultaneous search of the VA trellis using less than the maximum number of alternatives for each character and the trie representation of a dictionary can be achieved with an $r \times m$ Boolean array A, where the elements of A and the trellis nodes are in one-to-one correspondence. Element $A[j, i]$ is set to true if L_j is a possible correction for X_i, and false otherwise. One way of initializing A is by examining the letter confusion probabilities and setting $A[j, i]$ to true only if the probability of recognizing L_j as X_i is greater than a predetermined threshold. If a fixed number d of alternatives is used for each X_i, A is initialized by setting $A[j, i]$ to true only if the probability of recognizing L_j as X_i is among the d best choices for $1 \leqslant j \leqslant 26$. Therefore the paths of the trellis that need to be evaluated are only those that begin at the true-elements of the first column of A and proceed through the true-elements of the subsequent columns. Before evaluating

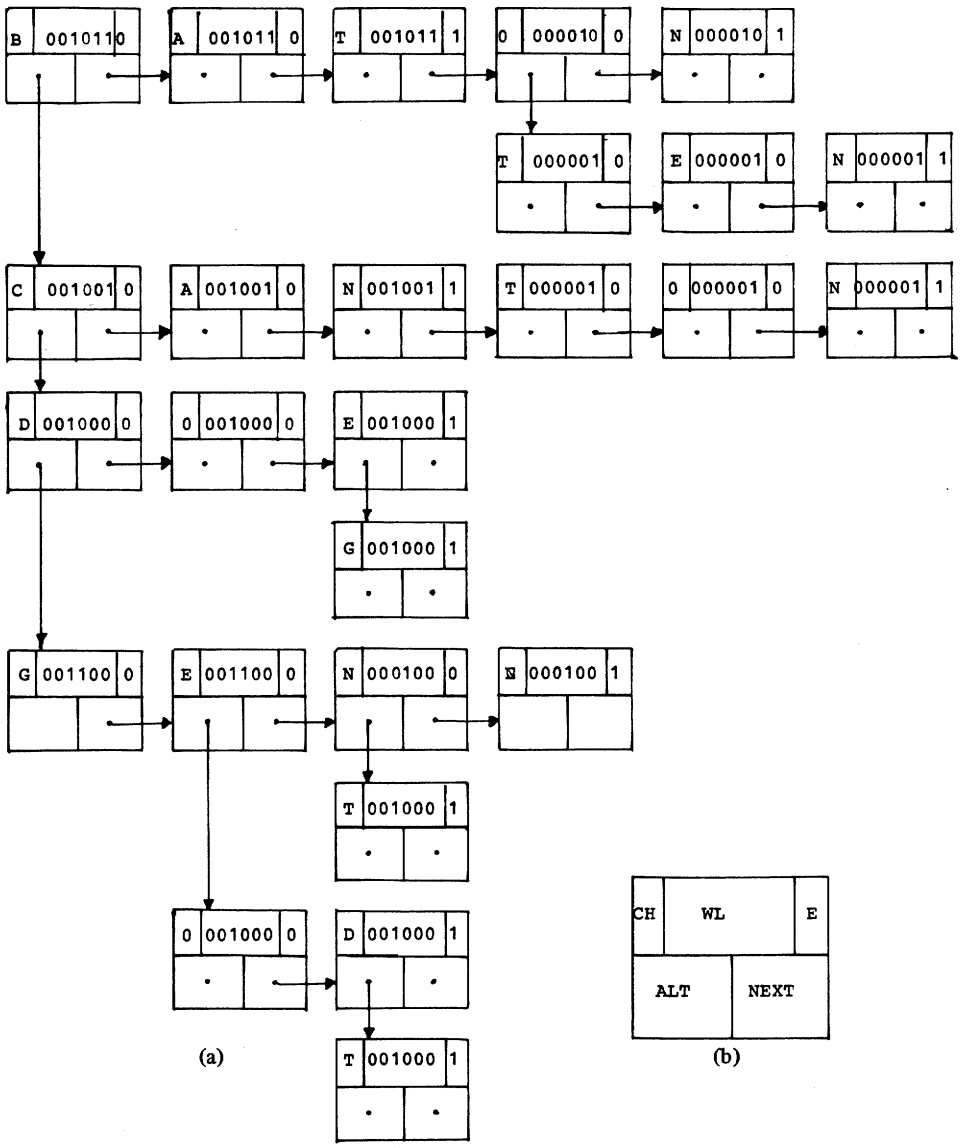


Fig. 2. (a) An example trie for the dictionary BAT, BATON, BATTEN, CAN, CANTON, DOE, DOG, GENE, GET, GOD, GOT. (b) The fields of each node.

a path that proceeds from one column of A to the next column, that path is determined to be legal with respect to the trie.

If $A[j, i]$ = true for $1 \leqslant j \leqslant r$, $1 \leqslant i \leqslant m$, i.e., all r alternatives are considered for each of the m letters, then the above process guarantees that the resulting string will be the most likely word of the dictionary. The optimality is because the VA guarantees the optimal string and the only paths of the trellis ignored at each stage are those that do not correspond to prefixes of words in the dictionary.

A formal statement of the main algorithm, procedure TEXT_RECOGNIZE, follows. This and other algorithms are stated in pseudo-Pascal with the convention that reserved words are boldface, procedure names are in uppercase, and variable names are in lowercase. The following type declarations are assumed global:

```
string = array [1 . . maxlen] of char; (* maxlen is the
                    maximum length of an input
                    word *)
    alpha = 1 . . r; (* r is number of characters in
                    alphabet *)
    length_in = 1 . . maxlen;

procedure TEXT_RECOGNIZE (check: Boolean);
var
  X, Z: string;
  m: length_in;
begin
  repeat
    GETWORD (X, m);     (* read next word X of m
                        letters *)
      if check
        then
          if INTRIE (X) (* look up X in the trie *)
            then
              Z: =X
            else
              DVA (X, m, Z)
```

```
        else
            DVA (X, m, Z);
    WORD_OUT (Z)    (* output word Z *)
  until end-of-file
end; (* TEXT_RECOGNIZE *)
```

TEXT_RECOGNIZE can operate in check mode (check = true) in which case the DVA is run only on input words that are not in the trie. This is because when an input word is in the dictionary it will most likely be output by the DVA, therefore the computation will be unnecessary. However for strict optimality TEXT_RECOGNIZE should be run with check = false.

A formal statement of procedure DVA follows:

```
procedure DVA ( X : string ; m : length_in; var Z : string);
(* given X as input, produce Z as output *)
var
  A: array [alpha,length_in] of boolean ;
  survivor : array [alpha] of string;
  cost : array [alpha] of real ;
begin
  INITIALIZE (A);
  DICTIONARY_TRACE (A,X,m,survivor,cost);
  SELECT (A,survivor,cost,Z)
end; (* DVA *)
```

Procedure INITIALIZE sets the elements of A to true that correspond to legal choices for each input letter, all other elements are set to false. Either a fixed number of alternatives or a variable number of alternatives can be used for each letter as described above.

Procedure DICTIONARY_TRACE, to be described next, traces through the trellis using the trie and the A matrix. It uses the vectors survivor and cost to hold up to r initial substrings of dictionary words and their costs:

```
procedure DICTIONARY_TRACE (A : array [alpha,length_in]
            of boolean; X: string; i: length_in;
            var survivor : array [alpha] of string;
            var cost : array [alpha] of real );
var
  j,k : alpha;
  oldcost, newcost: array [alpha] of real;
  oldsurvivor: array [alpha] of string;
begin
    if i > 1
      then
        begin
          DICTIONARY_TRACE (A,X,i-1,survivor,cost);
          oldcost := cost;
          oldsurvivor := survivor;
          for j := 1 to r do
            begin
              for k := 1 to r do
                if A[k,i-1] and A[j,i]
                    and STRING_IN_TRIE
                        (oldsurvivor[k],Lj)
                  then
                    EVALUATE_COST(oldcost[k],
                      Lk,Lj,Xi,newcost[k])
                  else
                    ELIMINATE_COST(newcost[k]);
              CHOOSE_SURVIVOR_COST(j,newcost,
                        oldsurvivor,survivor,cost)
            end;
        end
      else (* i = 1 *)
        INIT_SURVIVOR_COST(A,survivor,cost)
end; (* DICTIONARY_TRACE *)
```

DICTIONARY_TRACE uses i is as a pointer to the characters of the input word as they are processed. A formal description of procedure INIT_SURVIVOR_COST, that is called when $i = 1$, follows. The logarithms of confusion probabilities and first-order transition probabilities are assumed to be available in two $r \times r$ global arrays p_confus and p_trans:

```
procedure INIT_SURVIVOR_COST (A: array
                [alpha, length_in]
            of boolean; var survivor: array
                [alpha] of string;
            var cost: array [alpha] of real );
var
  j : alpha;
begin
  for j := 1 to r do
    if A[j,1] and STRING_IN_TRIE (null,Lj)
      then
        begin
          survivor[j] := Lj ;
          cost [j] := p_trans[Lj,b] + p_confus[Lj,X1]
        end
end; (* INIT_SURVIVOR_COST *)
```

As i progresses from 2 to m, DICTIONARY_TRACE determines the legality of the path in the trellis corresponding to the path from $A[k, i-1]$ to $A[j, i]$ according to these values and the result returned by the Boolean function STRING_IN_TRIE. STRING_IN_TRIE (directly analogous to the previously described ACCESS_TRIE) is given the survivor string corresponding to $A[k, i-1]$ (oldsurvivor[k]) and the letter corresponding to $A[j, i]$ (L_j). If there is an initial substring of a word in the trie consisting of oldsurvivor[k] concatenated with L_j, STRING_IN_TRIE returns true, otherwise false. When a legal path is found its cost is determined by procedure EVALUATE_COST that adds the cost of the transition from the node corresponding to $A[k, i-1]$ (given in oldcost[k]) to the node corresponding to $A[j, i]$ and returns the result in newcost[k], i.e., newcost[k] := oldcost[k] + p_trans[L_j, L_k] + p_confus[L_j, X_i]. When a legal path is not found, the trellis node corresponding to $A[k, i-1]$ is eliminated from consideration by assigning an arbitrary value of minus infinity to newcost[k] with procedure ELIMINATE_COST.

After all possible paths to the trellis node corresponding to $A[j, i]$ have been evaluated by the above method, procedure CHOOSE_SURVIVOR_COST determines the survivor for this node by choosing the path with the highest value in newcost. This value is returned in cost[j] and the new survivor is returned in survivor[j] as the appropriate element of oldsurvivor concatenated with L_j.

Procedure SELECT, which is called from DVA, returns the output string Z from the available words by choosing the one with the highest probability of being followed by a blank.

The simultaneous search of the trie and the VA trellis allows the number of possibilities to consider at each step to be constrained to prefixes of legal dictionary words. This is where the optimality of the algorithm follows from since when all r possibilities are considered for each letter, the algorithm is guaranteed to produce the dictionary word with the maximum *a posteriori* probability. However, this is usually unnecessary since it has been shown that values of d or t much less than their maximum produce the best correction performance [20]. This is because there are usually only a few viable alternatives for each character, thus the consideration of all possibilities is unnecessary.

In some cases, the use of values of d or t less than their maximum may cause input words to be rejected by the algorithm. This occurs when a set of paths is being traced through the trellis and the trie and at some point all the active trie paths contain letters that are not included in the trellis. Here the algorithm is unable to make a decision and a reject string is returned by CHOOSE_SURVIVOR_COST. This phenomenon is particularly pronounced with values of d or t less than those that provide the best correction performance since the correct alternatives for each character may not be included in the trellis. Increasing d or t past these settings is guaranteed to eliminate rejections but is more likely to produce an incorrect correction.

IV. A Cascaded Algorithm

The predictor-corrector algorithm (PCA) [18], [21] mentioned earlier is an example of a cascaded bottom-up and top-down approach to text recognition. The PCA processes an input word by first running the VA on it. The word output by the VA is then input to the dictionary-based portion of the algorithm.

A formal statement of the PCA follows:

```
procedure PCA ( X : string; m : length_in; var Z : string);
var
   center : 1 . . N[m] ; (* N[m] = number of words in
                             subdictionary of m-letter
                             words *)
   Y: string;
begin
     VA (X,m,Y); (* predict Y using Viterbi algorithm *)
       if IN_SUB_DICT (Y, m, center)
          then
             Z:=Y
          else
             CORRECT (Y,m,center,Z);
end; (* PCA *)
```

The PCA assumes a dictionary that has been partitioned into subdictionaries by word length where each subdictionary is arranged as a vector of words and the *value* of each word (the sum of its letter transition probabilities) has been computed and stored with it and the subdictionaries have been sorted by value. Under the assumption that when the values of two words are equal, the words are the same, IN_SUB_DICT determines whether a word is in the subdictionary of length m words by computing its value and executing a binary search by value. As a side effect it returns the index of the word with the closest value in center.

The dictionary is organized in this manner so that when an incorrect word is produced by the VA, center can be used by the top-down portion of the algorithm (procedure CORRECT) to define the center of a neighborhood in the subdictionary of m-letter words in which to search for the correct word. Presumably the constituents of this neighborhood and the incorrect word have similar values and it is likely that the correct word is among them.

The neighborhood considered by CORRECT is included between the indices lower and upper in the subdictionary of m-letter words. Lower and upper are defined below where f is a *fractional coefficient* that ranges between 0 and 1, $\lceil n \rceil$ is the ceiling of n, and $N[m]$ is the number of words in the subdictionary of m-letter words:

$$\text{lower} := \text{MAX}(1, \text{center} - \lceil f * N[m]/2 \rceil)$$
$$\text{upper} := \text{MIN}(N[m], \text{center} + \lceil f * N[m]/2 \rceil).$$

The *score* of each of the words that fall in this range is computed as its value plus the sum of the log confusion probabilities between its letters and those of the input word. The output is the dictionary word with the largest score.

The refinements of procedure CORRECT are below where WINDOW_SIZE (m) returns $\lceil f * N[m]/2 \rceil$ that has been computed *a priori;* GET_VALUE (score, m, i) places the value associated with the ith word in the subdictionary of m letter words in score $[i]$; GET_DICTIONARY_WORD (W, m, i) places the ith word in the subdictionary of m-letter words in W; and MAXWORD (score, lower, upper, Z) places in Z the word with the maximum score that falls between lower and upper. The input and output words are Y and Z, respectively.

```
procedure CORRECT (Y: string; m: length_in; center :
                                1 . . N[m] ;
                                var Z: string);
var
   i, j, lower, upper : 1 . . N[m] ;
   score: array [1 . . N[m] ] of real ;
begin
      lower := MAX(1, center-WINDOW_SIZE(m));
      upper := MIN(N[m] , center+WINDOW_SIZE(m));
         for i := lower to upper do
            begin
               GET_VALUE (score, m, i);
               GET_DICTIONARY_WORD (W, m, i);
                  for j := 1 to m do
                     score[i] := score[i] +p_confus [Y_j, W_j]
            end;
      MAXWORD (score, lower, upper, Z)
end ;
```

The PCA attempts to reduce the complexity of a dictionary method by using a heuristic (f) to limit the number of dictionary words for which computation is performed. It has been shown in [18] and in Section VI that as the heuristic is set at less than its maximum, correction performance deteriorates rapidly. This suggests the invalidity of the heuristic since it does not reduce computation without decreasing performance. This may be because of an inappropriate choice of dictionary arrangement, since the value of the correct word may bear no resemblance to that of the input word. This phenomenon and a method of counteracting it are discussed in [4].

The following section presents a theoretical comparison of the space and time complexity of the DVA and the PCA.

V. Computational Complexity

This section quantifies and compares the space and time complexity of the DVA and PCA. Storage is expressed as the number of memory locations (bytes) needed. Execution time is computed as the number of comparisons and additions, as a convention, both of these elementary operations are assumed to require equal units of computation [10].

Complexity of the DVA

The DVA uses a fixed amount of storage for confusion and transition probability tables, denoted as T_d bytes, and an amount of storage for the trie. The latter will be developed here as a function of the common word prefixes in a dictionary.

The lower bound on the number of trie nodes is the number of characters in the longest dictionary word and the upper bound is the number of characters in the serial dictionary. The actual number of nodes is a function of the number of letters in the original dictionary and S, the degree of sharing for a node in the trie. The degree of sharing is defined as the number of dictionary words that utilize the node to represent one of its characters. It can range between one and the number of words in the dictionary. Since every character in the dictionary corresponds to one and only one node in the trie, the degree of sharing for a character is defined as the degree of sharing for its corresponding node.

If there are N_c individual characters in the dictionary (numbered $1, 2, \cdots, N_c$) and if the degree of sharing for each character is given by S_i, then the number of nodes in a trie is given by

$$\sum_{i=1}^{N_c} \frac{1}{S_i} \tag{6}$$

and the number of bytes needed for the trie is the product of (6) and B, the number of bytes needed to represent a node.

The memory needed by the DVA is then expressible as

$$T_d + B \sum_{i=1}^{N_c} \frac{1}{S_i}. \tag{7}$$

A derivative of an expression representing the number of comparisons and additions needed by a formulation of the DVA analogous to that given for the VA in [18] for a fixed word length m and a fixed number of d alternatives was presented in [20] as

$$D(m, d, \tau) = 26m + m \sum_{j=1}^{\min(26-d, d)} (26 - j) + d^2(m - 1)(\tau + 3) + d(\tau + 1) + (2d - 1) \tag{8}$$

where τ is the number of comparisons used in a call to the function that determines the validity with respect to the trie of a string that begins at a given node in the trie. Its magnitude is determined by the number of alternate descendants at this node (see description of this function in Section III).

If the DVA is operated in the check mode then its computational complexity can be expressed as a function of α, the proportion of incorrect words input as

$$D_c(m, d, \tau, \alpha) = t_m + \alpha D(m, d, \tau) \tag{9}$$

where t_m is the number of comparisons necessary to "look up" an m-letter word in the trie.

Complexity of the PCA

The bottom-up portion of the PCA requires that storage be reserved for transition and confusion probability tables. It is assumed that T_p bytes are used for this purpose.

The top-down portion of the PCA uses a serial dictionary organization where each word has an explicit representation and an associated real-valued quantity. If the dictionary is partitioned into subdictionaries by word length m where there is $N[m]$ words in each subdictionary and if each character uses a single byte, each real quantity R bytes, and there are N_c characters in the dictionary, then the number of bytes needed by the PCA is given by

$$T_p + N_c + R \sum_{m=1}^{\max\_m} N[m] \tag{10}$$

where max_m is the length of the longest dictionary word.

The computational requirement of the PCA is a function of the computational requirement of the VA, the computation needed to determine value of the word output by the VA, to search the dictionary for the word with the nearest value, and to locate the word with the lowest sum of letter confusion probabilities.

The computational requirement of the MVA for a fixed word length m and a fixed value of the depth of search parameter d is given in [19] as

$$V(m, d) = 26m + m \sum_{j=1}^{\min[26-d, d]} (26 - j) + 3d^2(m - 1) - d(m - 4) - 1 \tag{11}$$

in the general case when $m > 1$ and $1 < d < 26$.

The computation of the value of the VA output requires $m - n + 1$ additions when nth order transition probabilities are used. The search to find the closest match to this value within the subdictionary of $N[m]$ m-letter words, needs $\lfloor \log_2 N[m] \rfloor + 1$ comparisons when a conventional binary search is used.

The number of units of computation needed for the selec-

tion of the word with the maximum sum of confusion probabilities is given in [18] as the product of the heuristic f and the units of computation needed for the maximization over the entire subdictionary. Since this portion of the algorithm is only executed for the percentage β of incorrect words output by the VA, the computation used by the PCA to process a word of length m can be expressed as

$$P(m,d,f) = V(m,d) + m + \lfloor \log_2 N[m] \rfloor + 1 + \beta f((m+1)N[m] - 1) \quad (12)$$

where $f((m+1)\,N[m] - 1)$ units are used to determine the word with the maximum sum of confusion probabilities. Although (12) is a linear function of the subdictionary size $N[m]$, the f heuristic allows this effect to be reduced.

Comparisons

The amount of storage needed by the DVA and PCA for probability information T_d and T_p, respectively, is a fixed constant that is usually negligible.

Due to the degree of sharing given in (6), usually the number of nodes in the DVA trie will be less than the number of characters in the PCA dictionary, but because each trie node needs more storage than a single character, the DVA trie may require more memory than the PCA dictionary. However, as more words are added to the dictionary the ratio of the number of trie nodes to the number of characters can be expected to decrease. This implies the existence of a cutoff point where the addition of more words to the dictionary will cause the serial representation to use more space than the trie. This phenomenon is discussed in the following example.

Let the nodes of the trie be represented as described in Section III using Pascal packed records. We store each record in a single word on a CDC Cyber 730 (which has a wordlength of 60 bits) using the following format: six bits for each character, 20 wordlength indicator bits, one end-of-word bit, and two 16-bit pointers (assuming a maximum of 2^{16} nodes). In implementing the PCA on the same machine, we can pack either a dictionary word of length between one and ten characters or a real value into a single 60-bit word. A dictionary word containing 11-20 characters will require an additional computer word. Let the longest dictionary word have 20 characters. When the number of nodes in the trie is less than the sum of twice the number of words with one to ten characters and three times the number of words with 11-20 characters, the above representation of the trie for the DVA will require less storage then that required by the PCA for its dictionary and value lists.

Comparison of the computational requirements of the DVA (8) and the VA (11) shows that the inclusion of top-down knowledge directly into the bottom-up processing of the VA does not change the order of complexity, i.e., it remains d^2. However, the coefficient of d^2 is increased by the trie search factor τ. Although the upper bound on τ is large ($4r$ for and alphabet of size r) [20] experimental results suggest a more reasonable value may be assumed. Furthermore, due to the linear dependence of the complexity of the PCA (12) on dictionary size, the inverse effect of increasing dictionary size on the increase in the number of trie nodes, and the fixed amount of computation for $V(m, d)$ in (12), it can be expected that although the time required by the PCA may be less than that for the DVA for small dictionaries, at some point the PCA execution time will exceed that of the DVA; the experimental results of the next section clarify some of these observations.

VI. Experimental Results

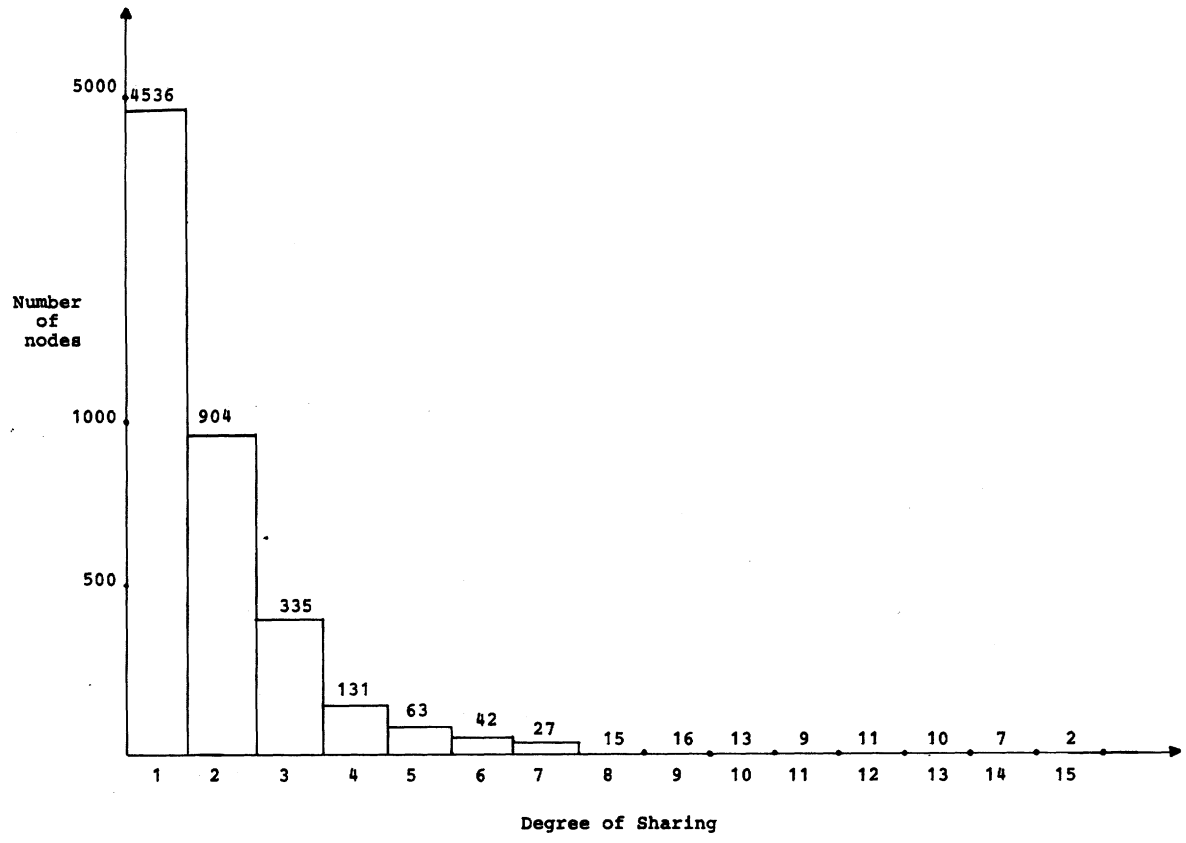
To empirically evaluate the performance and efficiency of the DVA and PCA, as well as the VA, in correcting character substitution errors, a database of correct and garbled text was established. The database of correct text consisted of 6372 words of English text in the computer science domain [26]. Bayesian estimates of single letter and first-order transition probabilities were computed from this text and a dictionary of 1724 words containing 12 231 distinct letters was extracted from it; the size of this dictionary is reasonable for a more general text recognition system, since it has been noted [12] that 1724 words can summarize 75 percent of written English.

A trie representation of the dictionary was constructed for use by the DVA. The parameters necessary to determine the storage and time requirements of the DVA, i.e., the degree of sharing and the number of alternate descendants, were estimated for the 6197 nodes in the trie. The results are shown in Fig. 3 where the histogram for the degree of sharing is extremely skewed with 73 percent of all nodes representing only a single character and 15 and 5 percent of the nodes representing only two and three characters, respectively.

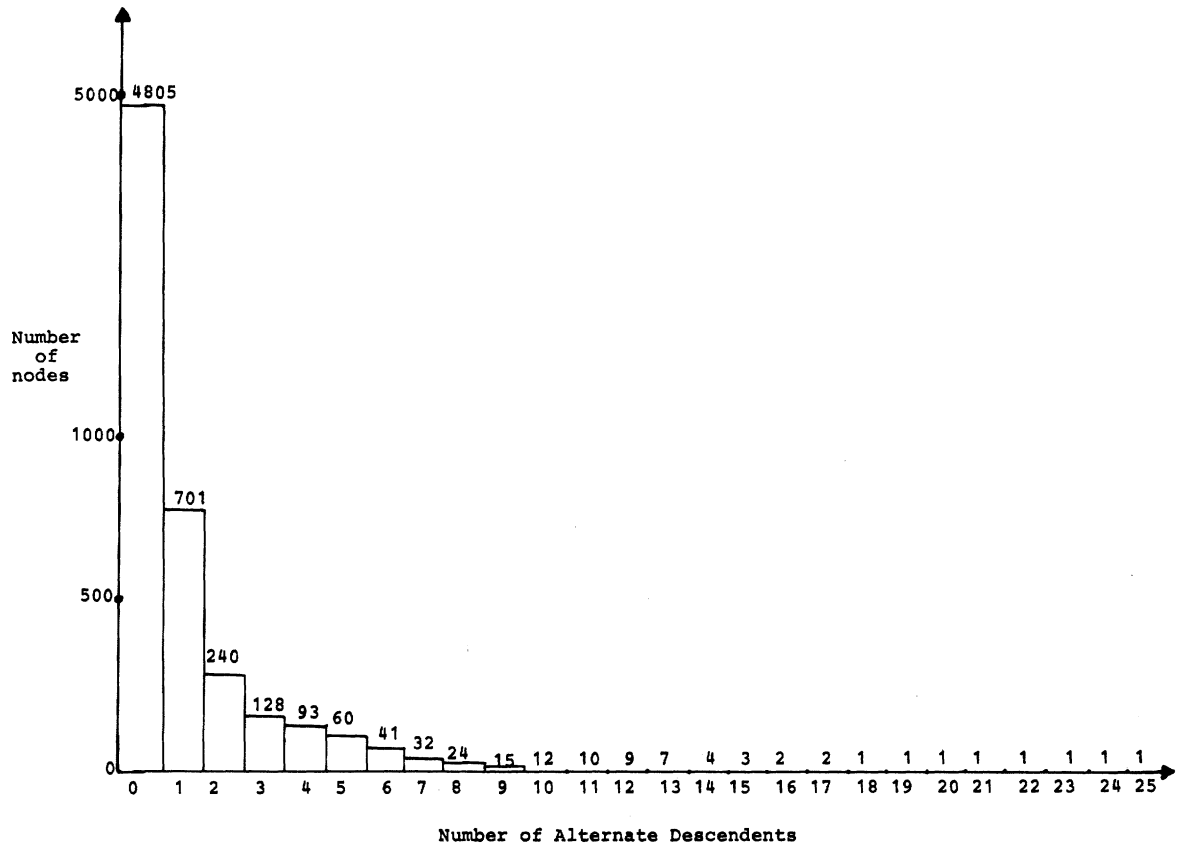
The histogram for the number of alternate descendants is also extremely skewed with 78 percent of the nodes having no alternate descendants and 11 percent of the nodes having one alternate descendant. This gives an indication of the effect of the trie access complexity factor τ on the computational complexity of the DVA (10). Since if at any given time there was a uniform probability of accessing any trie node, in 78 percent of all cases only a single node would have to be considered, i.e., much less than the upper bound of 26 nodes.

To test the correction capability of the three algorithms, garbled text with substitution errors was generated from the correct text using a model of a communications channel. A 31 percent word error rate was introduced into the correct text where 81 percent of these were single error words, 16 percent were double error words and 3 percent were triple error words. The confusion probabilities needed by all three algorithms were estimated from the garbled text.

The parameter settings of the PCA that yield optimum performance were estimated by running it on the entire garbled text. The results are summarized in Table I where a variable and a fixed number of alternatives are used by the VA phase of the PCA. The threshold parameter t was varied from -4 to -10 and the fixed number of alternatives parameter d was varied from 1 to 9. Within each setting of t and d the f heuristic was varied from 0.2 to 1.0 in increments of 0.2. Correction performance is given as the percentage reduction in word error rate between the garbled text and the text corrected by the PCA. From Table I it is seen that the best correction rate is reached when $t = -9$ and $d = 6$. This differs from the results of



(a)



(b)

Fig. 3. Degree of sharing and number of alternate descendents for nodes in the trie. (a) Number of nodes versus degree of sharing for each node. Only the first 15 degrees of sharing are shown. (b) Number of nodes versus number of alternate descendents for each node.

TABLE I
PCA PERFORMANCE FOR DIFFERENT SETTINGS OF THE f, t, AND d PARAMETERS. FIGURES ARE PERCENTAGE REDUCTION IN ERROR BY THE PCA FOR THE INDICATED PARAMETER VALUES. (A) VARYING f AND t. (B) VARYING f AND d.

	-4	-5	-6	-7	-8	-9	-10
0.2	16	16	25	42	50	51	51
0.4	28	28	37	53	61	63	63
0.6	39	39	47	64	72	74	74
0.8	47	47	54	71	78	79	79
1.0	56	56	62	76	82	83	83

(A)

	1	2	3	4	5	6	7	8	9
0.2	17	38	44	47	48	51	51	51	51
0.4	29	52	57	61	62	62	63	63	63
0.6	40	63	68	72	73	74	74	74	74
0.8	48	70	74	78	79	79	79	79	79
1.0	57	75	79	82	82	83	83	83	83

(B)

[18] where $d = 2$ was optimum. It is also seen that the optimum correction rate is reached only when the f heuristic is set to its maximum and that for any lower setting, correction performance drops off rapidly.

The above observations about the PCA were used in conducting further experiments to compare the performance of all three algorithms (DVA, PCA, and VA). The same garbled text and probability tables were used, the d and t parameters were varied as above, and the f heuristic of the PCA was set to its maximum. The results are summarized in Table II where the VA and PCA both reach their maximum correction rates of 30 and 83 percent, respectively, at $t = -9$ and $d = 6$. This is in contrast to the DVA that has a maximum correction rate of 87 percent reached at $t = -10$ and $d = 8$. It is also observed that the correction performance of the DVA is not influenced when the algorithm is run in check mode. However, the DVA in check mode requires only 40 percent of the time needed when it is not in check mode; also the time needed in check mode is even less than that needed by the VA that does not use a dictionary.

Another observation concerns the effect of different methods of estimating transition probabilities. It has been suggested [22] that Bayesian estimates be used to avoid taking the logarithm of zero. This assigns a small but finite probability of occurrence to all letter combinations and does not allow the use of information about letter combinations that never occur, e.g., 42 percent of all possible letter-pairs never occur in a large sample of English text [6]. In previous experiments with the VA [8] and the DVA [20], maximum likelihood estimates, that use an arbitrary value of negative infinity to represent the logarithm of zero, have been used. With the same databases of original and garbled text, a 35 percent correction rate was attained with the VA using maximum likelihood estimates [20], which is better than the 32 percent rate with Bayesian estimates described in this paper. This suggests that

TABLE II
EXPERIMENTAL COMPARISON OF MVA, PCA, AND DVA. TIME FIGURES ARE CPU SECONDS ON A CDC CYBER 730. (A) COMPARISON OF THE MVA, PCA, AND DVA FOR DIFFERENT VALUES OF THE DEPTH OF SEARCH (d). (B) COMPARISON OF THE MVA, PCA, AND DVA FOR DIFFERENT VALUES OF THE THRESHOLD (t).

	MVA		PCA		DVA			
					w/search		w/o search	
d	% error rate red.	time	% error rate red.	time	% error rate red.	time	% error rate red.	time
1	1	656	57	655	1	295	0	700
2	18	698	75	790	39	322	38	766
3	24	724	79	812	61	359	61	870
4	28	768	82	853	74	406	74	990
5	29	826	82	911	81	465	81	1154
6	30	893	83	977	85	536	85	1353
7	30	966	83	1051	86	629	86	1616
8	39	1057	83	1140	87	735	87	1925
9	39	1156	83	1240	87	873	87	2135

(A)

	MVA		PCA		DVA			
					w/search		w/o search	
t	% error rate red.	time	% error rate red.	time	% error rate red.	time	% error rate red.	time
-4	0	472	56	570	0	219	-51	472
-5	0	461	56	559	0	214	-18	476
-6	7	460	62	555	11	212	6	480
-7	19	479	76	569	47	237	46	558
-8	28	554	82	638	76	310	76	743
-9	30	641	83	725	84	387	84	976
-10	30	824	83	911	87	576	87	1502

(B)

maximum likelihood estimates are preferable to Bayesian estimates for implementing dynamic programming algorithms such as the VA, and so also for algorithms based on the VA such as the DVA and PCA.

VII. SUMMARY AND CONCLUSIONS

An integrated algorithm that simultaneously uses both bottom-up and top-down information in the recognition of words that may contain character substitution errors has been compared with a previous cascaded algorithm that separates bottom-up and top-down processes. Theoretical comparison of the computational and storage needs shows that the time and memory requirements of the integrated approach will be lower than that of the cascaded method when a large database is used.

The issue of an integrated approach to problem solving versus a modularly separated one has also been explored. In language understanding research it has been observed that systems that integrate syntactic and semantic processes perform better than those that cascade the processes. This was confirmed in the text recognition problem since it was shown that an im-

plementation of an integrated method for text recognition achieves a higher correction rate while using less time than a cascaded approach.

ACKNOWLEDGMENT

The authors wish to thank Ms. B. Rutledge who prepared the manuscript. Thanks are also due to Addison-Wesley for their permission to use parts of *Artificial Intelligence* by P. H. Winston [26] in the experimentation.

REFERENCES

[1] R. A. Amsler, "Computational lexicology: A research program," in *Proc. Nat. Comput. Conf.*, Houston, TX, 1982, pp. 657–663.

[2] W. W. Bledsoe and J. Browning, "Pattern recognition and reading by machine," in *Pattern Recognition*, L. Uhr, Ed. New York: Wiley, 1966, pp. 301–316.

[3] R. J. Bobrow and B. L. Webber, "Knowledge representation for syntactic/semantic processing," in *Proc. AAAI-80 Nat. Conf. Artificial Intell.*, Stanford, CA, 1980, pp. 316–323.

[4] D. C. Bouchard and G. T. Toussaint, "Heuristic search methods for efficient use of dictionary information in text recognition," School Comput. Sci., McGill Univ., Tech. Rep. SOCS 80.5, May 1980.

[5] W. Doster and J. Schurmann, "An application of the modified Viterbi algorithm in text recognition," in *Proc. 5th Int. Conf. Pattern Recognition*, Miami Beach, FL, 1980, pp. 855–863.

[6] E. G. Fisher, "The use of context in character recognition," Ph.D. dissertation, Dep. Comput. Sci., Univ. Massachusetts, Amherst, 1976.

[7] G. E. Forney, Jr., "The Viterbi algorithm," *Proc. IEEE*, vol. 61, pp. 268–278, 1973.

[8] J. J. Hull and S. N. Srihari, "Experiments in text recognition with binary *n*-gram and Viterbi algorithms," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-4, pp. 520–530, Sept. 1982.

[9] R. L. Kashyap and B. J. Oommen, "An effective algorithm for string correction using generalized edit distances," *Inform. Sci.*, vol. 23, pp. 123–142, 1981.

[10] D. E. Knuth, *The Art of Computer Programming, Vol. 1: Fundamental Algorithms.* Reading, MA: Addison-Wesley, 1968.

[11] D. E. Knuth, *The Art of Computer Programming, Vol. 3: Sorting and Searching.* Reading, MA: Addison-Wesley, 1973.

[12] H. Kucera and W. N. Francis, *Computational Analysis of Present-Day American English.* Providence, RI: Brown Univ. Press, 1967.

[13] F. E. Muth and A. L. Tharp, "Correcting human error in alphanumeric terminal input," *Inform. Processing and Management*, vol. 13, pp. 329–337, 1977.

[14] D. L. Neuhoff, "The Viterbi algorithm as an aid in text recognition," *IEEE Trans. Inform. Theory*, vol. IT-21, pp. 222–228, 1975.

[15] J. L. Peterson, "Computer programs for detecting and correcting spelling errors," *Commun. Ass. Comput. Mach.*, vol. 23, pp. 676–687, 1980.

[16] W. S. Rosenbaum and J. J. Hilliard, "Multifint OCR postprocessing system," *IBM J. Res. Develop.*, vol. 19, pp. 398–421, July 1975.

[17] R. C. Schank, M. Labowitz, and L. Birnbaum, "An integrated understander," *Amer. J. Comput. Linguistics*, vol. 6, pp. 13–30, 1980.

[18] R. Shinghal and G. T. Toussaint, "A bottom-up and top-down approach to using context in text recognition," *Int. J. Man-Machine Studies*, vol. 11, pp. 201–212, 1979.

[19] ——, "Experiments in text recognition with the modified Viterbi algorithm," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-1, pp. 184–192, 1979.

[20] S. N. Srihari, J. J. Hull, and R. Choudhari, "Integration of bottion," *ACM Trans. Office Inform. Syst.*, vol. 1, pp. 68–87, Jan. 1983.

[21] G. T. Toussaint, "The use of context in pattern recognition," *Pattern Recognition*, vol. 10, pp. 189–204, 1978.

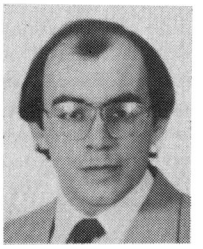
[22] G. T. Toussaint and R. Shinghal, "Cluster analysis of English text," in *Proc. IEEE Conf. Pattern Recognition and Image Processing*, Chicago, IL, 1978, pp. 164–172.

[23] T. N. Turba, "Checking for spelling and typographical errors in computer-based text," in *Proc. ACM SIGPLAN SIOGA Symp. Text Manipulation*, Portland, OR, June 1981, pp. 51–60.

[24] J. R. Ullmann, "A binary *n*-gram technique for automata correction of substitution, deletion, insertion and reversal error in words," *Comput. J.*, vol. 20, pp. 141–147, 1977.

[25] R. A. Wagner and M. J. Fischer, "The string to string correction problem," *J. Ass. Comput. Mach.*, vol. 21, pp. 168–173, January 1974.

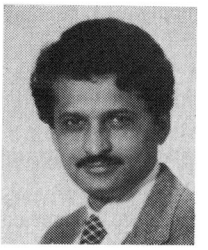
[26] P. H. Winston, *Artificial Intelligence.* Reading, MA: Addison-Wesley, 1977, ch. 9.



Jonathan J. Hull received the B.A. degree in computer science and statistics in 1980 and the M.S. degree in computer science in 1982, both from the State University of New York at Buffalo.

He is presently working on his doctoral dissertation on the topic of knowledge based text interpretation. His current interests are in the areas of text interpretation, pattern recognition, image processing, and computer graphics.

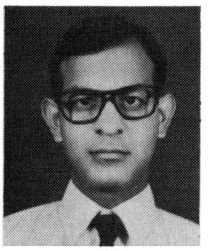
Mr. Hull is a member of the Association for Computing Machinery, the Pattern Recognition Society, the Association for Computational Linguistics, and the American Association for Artificial Intelligence.



Sargur N. Srihari (S'74–M'75) was born in Bangalore, India, on May 7, 1950. He received the B.E. degree in electrical communication engineering from the Indian Institute of Science, Bangalore, in 1970, and the M.S. and Ph.D. degrees in computer and information science from the Ohio State University, Columbus, in 1972 and 1976, respectively.

From 1976–1978 he was an Assistant Professor of Computer Science at Wayne State University, Detroit, MI. Since 1978 he has been with the Department of Computer Science at the State University of New York at Buffalo, where he is currently an Associate Professor. He has published over 30 research papers on the topics of pattern recognition, image processing, text interpretation/error correction, fault-tolerant computing, computed tomography, solid modeling, and computer graphics. He is currently editing a book *Computer Methods for Text Interpretation and Error Correction.*

Dr. Srihari is a member of the Association for Computing Machinery, the American Association for Artificial Intelligence, and the Pattern Recognition Society.



Ramesh Choudhari was born in India on March 25, 1950. He received the B.Sc. degree in physics from University of Bombay, the B.E. degree in electrical (electronics) engineering from the Indian Institute of Science, Bangalore, and the M.S. degree in computer science from the State University of New York at Buffalo.

He is presently an Assistant Professor of Computer Science at Claflin College, Orangeburg, SC. His research interests are in image processing, text processing, programming language design, and implementation.