Coding and Comparison of DAG's as a Novel Neural Structure with Applications to On-Line Handwriting Recognition

I-Jong Lin and Sun-Yuan Kung, *Fellow, IEEE*

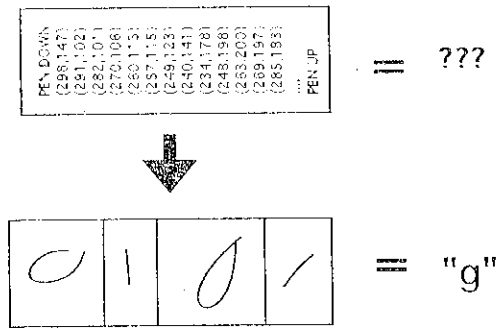
Abstract—**This paper applies directed acyclic graphs (DAG's) to a large class of (temporal) pattern recognition problems and other recognition problems where the data has a linear ordering. The data streams are coded (DAG-coded) into DAG's for robust segmentation. The similarity of two streams can be manifested as the path matching score of the two corresponding DAG's. This paper also presents an efficient and robust dynamic programming algorithm for their comparisons (DAG-compare). Since the DAG-coding methodology directly provides a robust segmentation process, it can be applied recursively to create a novel system architecture. The DAG structure also allows adaptive restructuring, leading to a novel approach to neural information processing. By using these elementary operations on DAG's, we can recognize on average 94.0% (writer-dependent) of the isolated handwritten cursive characters. DAG-coding may also be applied to speech recognition or any other continuous streams where a robust multipath segmentation aids the recognition process.**

I. Introduction

IN SPEECH and writing, the information is transmitted in a simple format: continuous streams, time-indexed series of real points. Reconstruction of these ideas or even words from the continuous stream is a difficult problem since a continuous stream has no obvious structure except for ordering. Structure is instrumental to the recognition process.

Continuous streams use ordering and locality to create *macrostructures* within themselves. Macrostructures are collectives of local and/or global elements within the continuous stream that define the location of "breaks" and consolidate sets of points into larger features. For instance, in Fig. 1, a series of points can represent a handwritten letter; these points can also be coalesced into a series of lines or loops. Through the ordering of these larger features, we can distinguish this constructed representation more readily than the original continuous stream.

Segmentation is an important step in recognizing these larger features: the process that identifies these macrostructures. Derived from the location of the macrostructures, "breaks" are inserted into the stream. In the directed acyclic graph (DAG) representation, these breaks are represented by *nodes*. The information contained between two "linkable" nodes, i.e., breaks, to be explained momentarily, is represented by an *edge*.



Example 1: Recognition of lowercase g

Nomanisanislandentireofitself

No man is an island entire of itself.
Example 2: Sentence Recognition

Fig. 1. Two examples that use segmentation for easier recognition. DAG's support robust use of segmentation for recognition.

Although there are many schemes for recognition problems such as time-dependent neural nets (TDNN), hidden Markov models (HMM's) [1]–[5], curve representation techniques [6]–[10], and some that use segmentation to some degree [11]–[14], our approach is fundamentally different in its treatment of segmentation. Traditional segmentation is a simple ordered set of breaks within the continuous stream and, moreover, links (edges) exist *only* between two consecutive breaks (nodes). These constraints lead to what we term a single-path DAG. In contrast, we believe that a multipath segmentation is vital for a robust recognition.

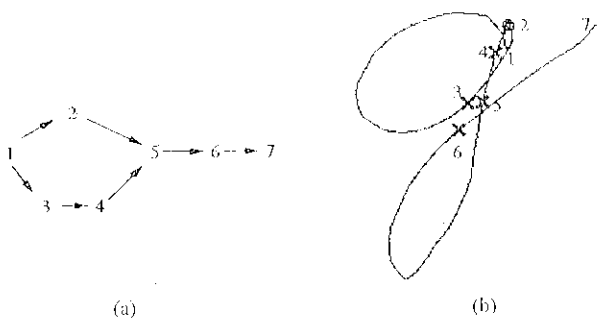
To support multipath segmentation, we express the segmented stream as a multipath DAG (see Fig. 2). DAG's compactly encode ordering information and ambiguity. DAG's also allow for a computationally efficient dynamic programming algorithm for robust comparison of these processed streams. By converting the continuous stream to a DAG, we also realize the adaptability of our data structure through graph transformations.

The coding of data streams as DAG's and the algorithm to compare DAG's present a high-quality solution to the problem of cursive handwriting recognition at the character level. We

Manuscript received June 17, 1997. The associate editor coordinating the review of this paper and approving it for publication was Prof. Jenq-Neng Hwang.

The authors are with the Department of Electrical Engineering, Princeton University, Princeton, NJ 08540 USA.

Publisher Item Identifier S 1053-587X(97)08119-1.



Theredraftisleaking.

(c)

(1) The (3) red (4) raft (5) is (6) leaking. (7)
There (2) draft

(d)

Fig. 2. (a) DAG structure for (b) a cursive lowercase G and (c) a stream of letters, (d) segmented at loop and cusp points and at recognizable word breaks, respectively. Macrostructures in the cursive letter are pen-lifts, pen-downs, cusps and loops; macrostructures in sentence are the valid word breaks.

Continuous Data Stream	$\rightsquigarrow$	DAG Structure
Breakpoint	$\Leftrightarrow$	A Node
Beginning of Data Stream	$\Leftrightarrow$	Source Node
End of Data Stream	$\Leftrightarrow$	Sink Node
Segment of the Data Stream	$\Leftrightarrow$	An Edge
Partial Ordering	$\Leftrightarrow$	Direction of Ares
Derived Data Structure from a Data Segment	$\Leftrightarrow$	Edge Value
Ambiguity, Duality	$\Leftrightarrow$	Multiple Path With DAG
Hypothetical Segmentation	$\Leftrightarrow$	Path from source to sink
All Hypothetical Segmentations	$\Leftrightarrow$	DAG

Fig. 3. Correspondence between DAG's and a data stream.

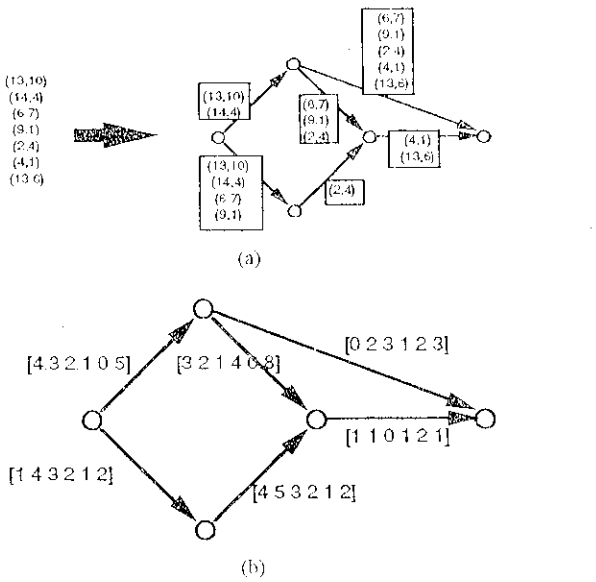


Fig. 4. Example of a continuous stream of points converted to a DAG with edge values as three-dimensional (3-D) vector. (a) Continuous stream after segmentation and (b) DAG-coded continuous stream after segmentation and edge value extraction.

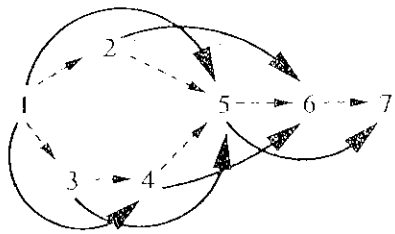


Fig. 5. DAG that deals with imperfect segmentation. Spurs are the breaks that are wrongly inserted into the continuous stream. With additional edges to bypass any number of a nonconsecutive spurs, this DAG can account for these spurs. The original edges are dashed; the solid, dark edges are added to account for a nonconsecutive spurs.

are of the opinion that *DAG's can be used on any continuous data stream where a robust segmentation process can aid in recognition, such as speech recognition.*

II. Extraction of Continuous Streams to DAG

A. Data Structure

Our basic data structure corresponds very closely to a polar DAG [15], which is a directed acyclic graph with two distinguished nodes (a sink and a source), but we add real-valued data structures on each edge called *edge values*. When we refer to a DAG in this paper, we refer to this particular flavor of DAG. A full correspondence between DAG's and a data stream is given in Fig. 3. An instance of a DAG expresses multiple traditional segmentations within a single structure.

Our multipath DAG differs fundamentally from a traditional single-path version used in dynamic time warping (DTW): Multipath DAG's integrate a complex segmentation process into its structure. Segmentation schemes insert a series of breaks into the continuous stream. Traditional schemes generate their breaks causally or locally, basing the location of breaks on previous or local information; they recognize segments of data only between breaks and their immediate predecessor. With these breaks and their simple dependencies, traditional segmentation schemes produce a single path DAG representation such as those used by DTW in speech recognition.

Multipath structures support complex constructs such as contextual segmentation schemes, ambiguous and dual structures, and segmentation fault models. The multipath structure of DAG's give segmentation schemes the freedom to create all ordered dependencies between generated breaks. Since dependencies between breaks are no longer constrained by a strict ordering, our multipath segmentation scheme can take a global perspective and incorporate more information in the generation of a break than what is available between two consecutive breaks.

B. DAG Coding

DAG coding combines the global structure from segmentation and local informational extraction from edge value extraction to provide a robust representation. A simple DAG coder takes a single continuous stream and yields one DAG

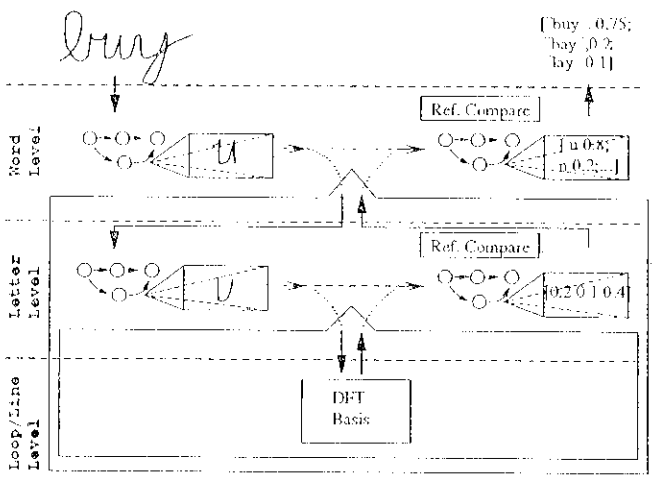


Fig. 6. Recursive architecture that uses DAG's.

For a specific example of a simple DAG coding, refer to Fig. 4. For the use of DAG coding in our application, see Section V-B.

1) *Segmentation*: Segmentation finds where the continuous stream is to be broken into segments and derives a DAG that encodes all possible sets of breaks and their respective segments. [See Fig. 4(a)] Segmentation schemes with a DAG can also combine different informational contexts to create complex feature structure. In our application for handwriting recognition, we combine spatial analysis with time indexing (see Section V-B).

   Segmentation also reduces the number of elements that represent information compared with the number in the original continuous stream. Unlike a single-path DAG, a multipath DAG can compensate for this reduction by creating multiple path structures. A single-path DAG implicitly classifies a segment of the continuous stream and forces ownership of the data segment on a single feature; a multipath DAG can put a data segment within multiple paths, allowing for multiple ownership of a data segment. When the definitions of macrostructures overlap, DAG's can express this duality through redundancy.

   Multipath DAG's can also compensate for imperfect segmentation. For instance, spurs are breaks that are wrongly inserted into the continuous stream. Segmentation for multipath DAG's can deal with nonconsecutive spurs by adding edges to bypass these spurs (see Fig. 5).

2) *Edge Value Extraction*: Edge value extraction represents the data between two breaks as a single representation. Edge value extraction collapses the data segment between two breaks into a single edge value under the assumption that only one macro feature exists between the two breaks. [See Fig. 4(b)]

3) *DAG Coding of Multiple Data Streams*: DAG coding can take more than one data stream as input. A simple DAG coder takes only one sample and produces one DAG; we can extend this concept of DAG coding to represent multiple inputs with one DAG. *The complex DAG coder produces a single DAG that generalizes multiple inputs with a single structure like a neural net.* However, we defer this discussion to Section IV.

C. Analysis of DAG's

Although any continuous stream can be DAG coded, it must have certain characteristics to benefit from representation in DAG's. This section discusses these characteristics and their implications on DAG's.

1) *Existence of Identifiable Macrostructures*: The segmentation process attempts to break up the continuous data stream with respect to *identifiable* macrostructures. If there are no such macrostructures within the continuous stream, segmentation has no meaning.

2) *Information Ordering*: DAG's require both the continuous stream and the macrostructures within the continuous stream to be ordered. Information ordering allows a path within DAG to be a compact representation of the continuous stream after segmentation. If elements for a single feature are not localized between two macrostructures, edge values cannot be extracted. If there is no strong ordering of the macrostructures, paths and partial paths within DAG's no longer represent segmentation, invalidating our dynamic programming comparison algorithm (see Section III).

3) *Segmentation Granularity*: Efficiency and quality of the DAG's solution is directly related the reduction of data in its extraction. If segmentation is too fine, DAG coding will have little advantage over the original stream since the number and ordering of elements will not change appreciably. If the segmentation is too coarse, conventional feature extraction through single functional mapping is a simpler solution than DAG's. In general, DAG coding needs to balance ordering information with edge value extractor complexity.

4) *Cost of Redundant Information*: The DAG multipath structure tends to represent the same information on different paths. The price of redundancy is hopefully compensated by improved recognition performance.

D. Recursive Structure

DAG coding and its complementary DAG compare operation are a divide-and-conquer method that is independent of level of language recognition. Thus, they form a robust inductive step within a structural recursion. By choosing another DAG coder and its DAG compare operation as an edge value extractor for a higher level DAG, a DAG coder for the edge can be considered to be the inductive step within a recognition system architecture (see Fig. 6).

Going into this recursive architecture, the system recursively applies the segmentation to divide the problem and uses the DAG to maintain dependencies among subproblems. The architecture recurses on subproblems until the subproblem is small enough for a simple basis function. Coming out of the recursive architecture, the recognizer repeatedly applies the DAG compare operation to consolidate the recognition results.

In cursive handwriting, breaks exist at different levels of representation: loop/line, character, word, and sentence level. Each level of representation corresponds to a level within the recursive architecture that is orthogonally designed and

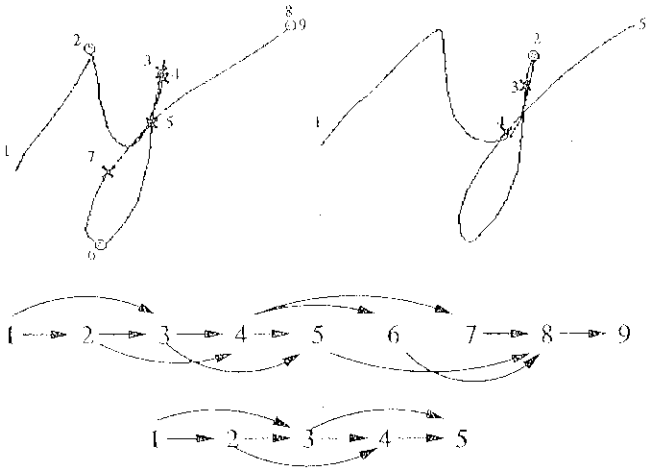


Fig. 7. Two segmented Y and their DAG's. Circles and X's correspond to breaks. Note that both Ys have very different segmentation and very different DAG structure, but they are scored as very similar when compared using our path matching algorithm. The left letter has the top DAG structure; the right letter has the bottom.

optimized. This structurally recursive architecture and its implications on recognizer design are discussed in detail in [16].

III. Algorithm for Matching DAG's

To take full advantage of the expressiveness of DAG's, this paper also presents an efficient algorithm to measure similarity of DAG's (DAG-compare). The path matching algorithm is a variant of the Viterbi that uses dynamic programming to find the best matching path between two DAG's. The algorithm's output is a real-valued similarity score for two DAG's. This score can be used for classification by comparing DAG-coded input to DAG-coded exemplars. *Similarity of two DAG's is defined as the highest path matching score of any two paths between the two DAG's.* (See Figs. 7 and 8.)

A. Path Matching Score

In this section, we will impose a minor restriction on the formulation of path matching score so that computation is tractable. If we allow the path matching score to be any arbitrary function (especially a boolean one), the problem can be NP complete. We restrict the path score to a particular recursive formulation with a requirement of monotonicity.

For our path matching score, we define a function MatchEdge within a recurrence relation that relates the previous path matching score to the next path score after traversing an edge for each path in each of the two DAG's. A matched edge is allowed to be null, allowing the DAG comparison to "skip" some edges.

$$MatchEdge(Edge\_1, Edge\_2, prev\_score) \rightarrow next\_score$$
$$s.t. \forall\{Edge\_1, Edge\_2\}, prev\_score \geq next\_score \tag{1}$$

The MatchEdge function compares the edge values and returns a score. A larger mismatch between edges should mean a lower returned score. MatchEdge may also accept null edges that correspond to the "skipping" of an edge.

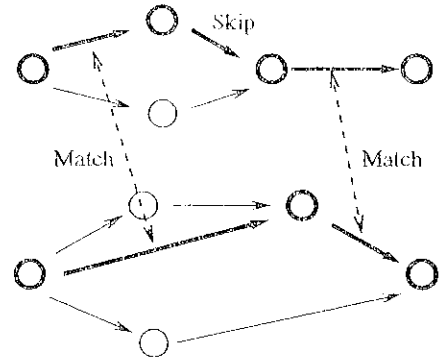


Fig. 8. Example of path matching with node and edge correspondence. Darkened nodes and edges correspond to the two paths that give the highest matching score. The dotted lines show which two edges were matched together.

The selection of MatchEdge is up to the designer but should obviously model the matching probability and informational loss associated with each edge value. To reduce computational complexity, MatchEdge must be monotonically decreasing with respect to the previous score, as defined in (1). However, a better match can mean less of a decrease in score. Only a perfect match can avoid any loss in score.

The basic definition of path matching is very similar to the DTW: an ordered correspondence between nodes and edges in the two paths that gives the highest score (see Fig. 8).

B. Dynamic Programming Path-Matching Algorithm for DAG's

This section presents a polynomial-time algorithm for path matching between two DAG's. The matching score between two paths can be solved in polynomial time [17]. However, a DAG comparison compares all pairs of paths between two DAG's, which requires an exponential number of path comparisons. Fortunately, dynamic programming can be applied to this problem to construct a polynomial time algorithm for comparison of two DAG's.

The problem is to compare all paths between source and sink of one DAG to all the paths between source and sink of another. Let us define the subproblems associated with this problem.

$$score(x, y) = \begin{cases} \text{Set of all path scores for one path that} \\ \text{goes from source and to node } x \text{ (in} \\ \text{one DAG) and another that goes from} \\ \text{source to node } y \text{ (in the another DAG)} \end{cases} \tag{2}$$

where we wish to find

$$\max(score(sink_{DAG1}, sink_{DAG2})) \tag{3}$$

The previous subproblems can be reached by traversing at most one edge in each graph to the present subproblem. In the DTW algorithm, the single path DAG structure only allows a node in the graph to be dependent on its previous node, simplifying the computation. Since our algorithm accepts a multipath DAG, the previous subproblems are not as regular

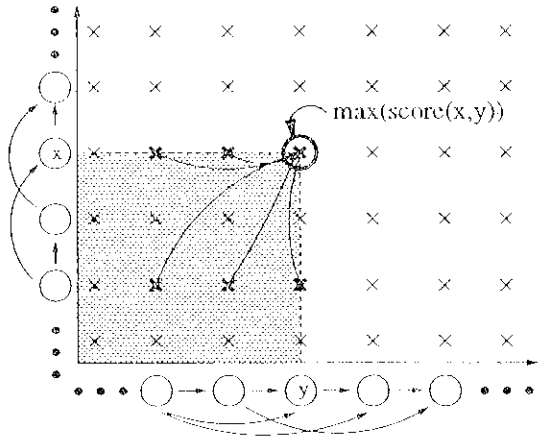


Fig. 9. Computational dependencies for dynamic programming algorithm for path comparison of two DAG's. The previous problems are represented by the crosses. Note that all previous problem triangles are contained within the shaded area.

as DTW but are defined in the same manner.

$$score(v_1,v_2) = \bigcup_{\substack{e_1 \in InEdges(v_1),\\ e_2 \in InEdges(v_2)}} \left\{ \begin{matrix} MatchEdge(e_1,e_2,\\ score(e_1.prev,e_2.prev)) \end{matrix} \right\}$$

$$\text{where } InEdges(v) = \{e | e.(x,v), e \in E\} \tag{4}$$

To find a maximum of this set, this formulation still leads to a possibly exponential time computation. By using MatchEdge's monotonicity from (1), we can simplify the equation for the maximum of a given score set.

$$\max(score(v_1,v_2)) = \max(S) \tag{5}$$

$$S = \bigcup_{\substack{e_1 \in InEdges(v_1),\\ e_2 \in InEdges(v_2)}} \left\{ \begin{matrix} MatchEdge(e_1,e_2,\\ \max(score(e_1.prev,e_2.prev))) \end{matrix} \right\} \tag{6}$$

By dynamic programming principle, we recognize that only the best score needs to be remembered at the previous step. All that remains is to be shown is that there exists an order of solving the maximum of the score sets that preserves these edge dependencies, i.e., all computation of previous scores can be finished before traversing another pair of edges.

In fact, there are many orders of solving the subproblems that preserve these dependencies. Since both graphs are DAG's, we can topologically sort [17] the nodes in each graph such that all edges leave from one node and lead to another of higher order. Thus, a subproblem is only dependent on other subproblems that are contained within the rectangular area between the problem and the origin. Graphically speaking, the inductive process of solving subproblems can be pictorially viewed as growing an area along the axes such that any point to be added to the area is supported both horizontally and vertically from each axis (see Figs. 9 and 10).

To find the path matching score of the two DAG's, the solution for subproblems are calculated until $\max(score(sink_{DAG1}, sink_{DAG2}))$ is found. To find the actual paths and the edge matching that resulted in this score, we employ a backtracking method similar to the Viterbi algorithm.

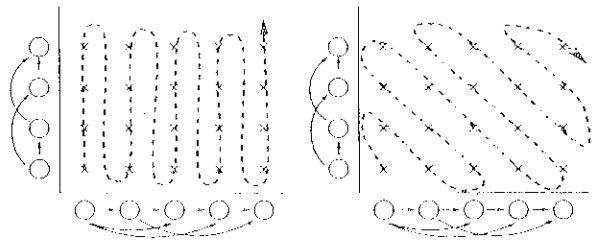


Fig. 10. Two valid orders of computation.

C. Proof of Optimality

The application of (5) and (6) in a valid traversal of subproblems finds the optimal (maximum) path matching score of all paths encoded between two DAG's with MatchEdge as defined in (3). Proof of optimality is given by a simple induction on the number of solutions for the maximum score sets. The basis is the path score before traversing any edges. For the inductive step, consider the $(n+1)$th subproblem. By the inductive hypothesis, all the previous n subproblems have been visited and have the optimal score for their subproblem. Since our traversal of subproblems is valid, all maxima of previous score sets on which the maximum for the $(n+1)$th score set is dependent have already been computed. By running this algorithm to the final maximum, we obtain the $\max(score(sink_{DAG1}, sink_{DAG2}))$, which is the path-matching score for two DAG's as defined in (3).

D. Running Time

If MatchEdge is an $O(1)$ operation, the running time of our algorithm is

$$O((|V_1| + |E_1|)(|V_2| + |E_2|)) \tag{7}$$

where (V_1, E_1) and (V_2, E_2) are the graphs of the two DAG's to be compared.

E. Comparison to Dynamic Time Warping

Our algorithm takes into account the freedom of connectivity, fusing the power of a fully generalized DTW-like algorithm on top of an adaptive structure. Our algorithm encompasses DTW in its definition [18] (see Fig. 11). DTW is limited by its localized connectivity in its computation dependencies. Furthermore, connections in DTW are still under the constraints of regularity of structure, whereas DAG comparison can take any two DAG's. Since DAG comparison can accept structures of irregular and dynamically varying connectivity, it can be applied to adaptive neural structures. Global path constraints in DTW can be accommodated with minor adjustments to the algorithm.

The drawback to our approach is that the freedom of connectivity within DAG comparison removes any regularity within design implementation, which is key to mapping onto array structures. This nonuniformity of structure makes our algorithm slower in implementation than a similarly sized DTW algorithm.

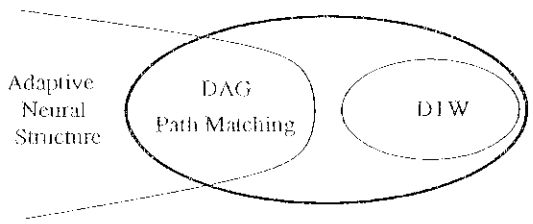


Fig 11 Algorithmic space of dynamic time warping (DTW) and DAG comparison DTW can be generalized by relaxing its locality constraints However DAG comparison fully generalizes the connectivity of the string comparison with its DAG structure Furthermore, this flexibility allows DAG comparison to be used on adaptive structures, just like a neural net

IV Neural Structure of DAG's

With its data structure defined and algorithmic framework in place, DAG's have another dimension: adaptive neural structure With their adaptability, DAG's can be coded to learn and represent multiple DAG coded input streams, i e , complex DAG coding This capability in DAG's allows for a novel learning algorithm that integrates parametric and structural optimization Unlike block-updating parametric optimization that depends on cumulative effects of each member of the training set, a DAG learning scheme optimizes structure by negotiating graph transformations with respect to the training set as a whole

A Memory DAG's

One of the simplest learning schemes is memory For DAG's (as for HMM), such an adaptive learning scheme is trivial: DAG coded outputs are collected into a parallel DAG network, and a DAG compare on this network results in the score of the most similar output that had been previously seen We use this simple but effective learning scheme in Section V However, unlike traditional schemes with fixed structure, we can use this "memory" DAG as a starting point for structural learning Just as the distillation of memory leads to experience, reducing the structure of this "memory" DAG will correspond to learning

B Strategies for Learning

Unlike block-updating schemes, DAG learning is inherently structural Although memory DAG's are a naive learning scheme, they still maintain a global view of the training set through its graph structure Since the structure of the memory DAG maintains visible dependencies among the members of the training set, graph transformations will take into account these interconnections Two forms of reductions can be applied to a DAG network: edge and node reduction Multiple edges that begin and end at reducible nodes can be clustered through traditional methods such as vector quantization, k-means, etc Nodes are reducible only if their successor or predecessor structures are also compatible Like graph network problems, there are well-defined expand and reduce transformations that can lead to the optimally distilled DAG The learning schemes for DAG's looks to be an exciting topic and is the focus of future work

C Algorithmic Convergence of ACON and OCON

DAGs' global view of learning extends beyond the scope of single-class training DAG's provide a spectrum of all-classes-one-net (ACON) and one-class-one-net (OCON) networks [19], where the degree of sharing among classes is governed purely by the network structure This spectrum of networks is based on the same algorithmic kernel [16] but manifests itself as changes in the structure of the memory DAG Just as DAG coded outputs for a single class may be combined into a single DAG, DAG coded outputs for multiple classes may be also combined into a single DAG with multiple sinks and be similarly reduced Clearly, we will also pursue the theoretical implications of structural training of DAG's

V Application to On-Line Cursive Handwriting Recognition

We now apply DAG's to a specific continuous-stream, time-sampled points of a pen from a digitizing tablet From these points, we can extract DAG's at loop/line level from sampled pen points of individual written characters This DAG coded representation of the data is compared to similarly DAG coded exemplars The matching score will determine the classification This experiment forms the basis of a cursive handwriting recognition system The system is user dependent and user trained Design of such a system through application of DAG's and experimental results are detailed below

A Cursive Handwriting as DAG Problem

Connected cursive handwriting is a good candidate for DAG's First, cursive handwriting at the word level is a point stream of characters The continuous stream of characters can also be seen as a stream of loops and line segments broken at cusps, loops, and the lifting of the pens Second, cursive handwriting strictly enforces its ordering by forcing the pen always to be touching the paper except for word breaks and the crossing of t's, etc The DAG coding of cursive handwriting at the loop/line level is relatively straightforward; since block printing does not have such a strict enforcing, a DAG solution would require a special ordering scheme In this section, we shall concentrate on handwritten cursive character recognition on the loop/line level and its respective segmentation

1) Problem Statement In this case, the input data is simply the time-sampled (x, y) coordinates of the pen position on the pen tablet sampled at approximately 45 Hz A special symbol is given when the pen is placed on or lifted off of the tablet Each cursive letter is given as a separate set of points to be recognized The problem is to recognize all letters for one writer, given only a known subset that contains example(s) of every letter

2) System Design The recognition system design is simple (see Fig 12) Known exemplars for each letter as well as test letters are DAG coded at the loop/line level Test letters were then compared to DAG coded exemplars The letter for test data is selected by choosing the letter of the highest scoring exemplar For this DAG solution instantiation, we must define the segmentation and edge value extraction along with our MatchEdge function

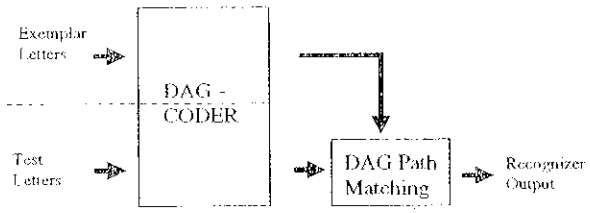


Fig. 12. Block diagram for cursive letter recognizer. Note that system is user-dependent and user trainable. There are no restrictions on how a writer draws a letter other than that the letters are reasonably consistent and recognizable by the writer.

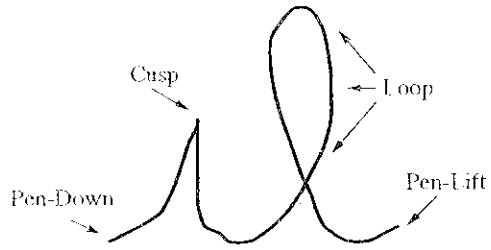


Fig. 13. Examples of the three types of macro-structures from left to right: Pen-down, cusp, loop and pen-lift.

B. DAG Coder for Cursive Letter Recognition

1) *Segmentation for DAG-Coding*: Segmentation at the loop/line level is relatively simple and has been described before in [14]. There are four types of macrostructures: loops, cusps, pen-down, and pen-lifts (see Fig. 13). Lines are not considered macrostructures since they do not create any breaks. Rather, they are products of a pair of breaks. On the other hand, we consider loops to be macrostructures. Once a loop is identified, two breaks are created: one at the beginning of the loop and one at the end. Cusps are another type of macrostructure: the points at which the differential of the curvature exceeds a certain threshold. A cusp breaks the stream in the middle of the cusp. Pen-downs and pen-lifts are when the pen first touches and leaves the writing surface and are assumed to be perfect information for the location of breaks. Our segmentation scheme also accounts for nonconsecutive spurs, as mentioned in Section II-B. Cusps may exist within loops and must expressed in terms of a dual path structure.

2) *Edge Value Extraction*: Edge value extraction is defined in Fig. 14. Edge value extraction at the loop/line level is a feature vector created from the low-frequency terms of complex discrete Fourier transform of the line segment position in the complex plane, relative y position, and relative distance that the pen travels within the segment. It can be shown that CDFT encodes direction, loop eccentricity, and how well the ends of line segment meet.

C. DAG Comparison for Cursive Letter Recognition

For our DAG comparison, the function MatchEdge is defined in (8)–(10) (see Fig. 15). Empirical results guide us for a preferable choice. In the case, the MatchEdge function is an exponential Gaussian dependent on the radial distance between the two edge values. The MatchEdge function also depends more on the shape of the line segment as the line segment becomes longer. Skipping deducts a cost proportional to the distance the pen travels for that segment. Since the score goes to zero after 25% of the information is skipped, the MatchEdge function disallows any result that does not use at least 75% of the available information. The path minimum matching score is 0; the maximum is 1.

Edge Value definition:

$E[1$	$8]$	Real Part of Low Freq. CDFT terms
$E[9$	$16]$	Imaginary Part of Low Freq. CDFT terms
$E[17]$		Relative distance that pen travels in segment
$E[18]$		Relative average height of pen within segment

Fig. 14. Edge value definition for cursive handwriting at the loop/line level.

$$Match(score, e_1, \lambda) = score - 4(e_1[17]) \quad (8)$$

$$Match(score, \lambda, e_2) = score - 4(e_2[17]) \quad (9)$$

$$\begin{aligned} &MatchEdge(score, e_1, e_2) = \\ &\quad score \times \exp\left(-\sqrt{(\vec{e_1} - \vec{e_2})^T C (\vec{e_1} - \vec{e_2})}\right) \\ &\quad \text{where } C = diag(d_1, \ldots, d_{16}, 8, 8) \\ &\quad \text{and } d_i = 1 + (e_1[17] + e_2[17])/2, 1 \le i \le 16 \end{aligned} \quad (10)$$

Fig. 15. MatchEdge definition for cursive handwriting at loop/line level.

D. Experimental Results

The DAG solution recognizes all lowercase letters not only with high accuracy (around 94%) that approaches human recognition rates but also with robust precision. The DAG solution places the correct letter within the top three choices >99% of the time on average over all ten writers, using only three exemplars per letter. The system is user dependent and user trainable. There were no restrictions on how a letter is drawn other than that the letters are reasonably consistent and recognizable by the writer. Furthermore, the design of recognition systems was relatively simple, computationally efficient, and extremely tolerant to variations in writing styles. The complete results are shown in Fig. 16. The source code is available at [16] and [20].

Handwriting samples were taken from ten different subjects. All lowercase letters from a to z were used. A subset of letters were used as exemplars for comparison. Each letter class was given the same number of exemplars, and the exemplars were chosen randomly for each run. The recognition program was run eight times with different sets of exemplars for the average value. The range column in Fig. 16 correspond to the worst and best single run for a particular writer. These results do NOT use any word-based or neighboring letter contextual information.

Recognition rates were normalized by relative frequency letters in the English Language [21]. The final results approach human recognition rates of discretized cursive handwriting of 95.6% [22]. A common recognition error is between lowercase cursive e and l. Since we enforce a strict isolation between characters, the two letters cannot be distinguished by relative height. However, if we were to add postprocessing to distin-

Number of Top Choices	Number of Exemplars per letter	Avg. Num Test Cases per letter	Recog. Rate Range	Recog. Rate Avg.
1	3	9.7	85.0-100%	94.0%
2	3	9.7	92.5-100%	98.2%
3	3	9.7	94.9-100%	99.1%
1	2	10.7	82.7-99.5%	91.9%
2	2	10.7	87.5-100%	97.1%
3	2	10.7	92.7-100%	98.7%
1	1	11.7	70.9-98.3%	86.3%
2	1	11.7	79.4-100%	93.8%
3	1	11.7	87.7-100%	96.5%

Fig. 16. Results of isolated cursive character recognizer that uses DAG's. Correct result is when the correct letter falls within top choices as defined in the first column in the table. Character recognition are normalized to letter probability of English language. *Results do NOT use neighboring character or word-level information.*

guish *e* and *l* by height, the first recognition rate for the three exemplars per letter coincidentally rises to 95.6%.

VI. Conclusion and Further Study

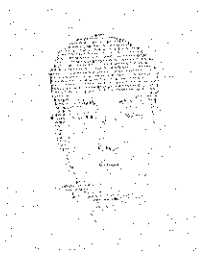
In this paper, we have presented a method that supports the process of segmentation by compactly encoding ordering and ambiguity within DAG's. We have presented a model of comparing DAG's and have shown a optimal polynomial-time method for comparing segmented continuous streams. The general theory of DAG's can be applied to many different types of recognition systems to provide a robust recognition solution.

Acknowledgment

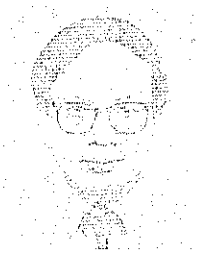
The authors would like to thank the people who contributed handwriting samples: J. Britting, M. Carroll, R. Dick, C. Kreatsoulas, C. Lee, D. Nawi, K. Purvis, K. Wang, D. Williams, and H. Xuereb. Special thanks to Prof. S. Malik for a timely bit of encouragement.

References

[1] J. R. Bellegarda, D. Nahamoo, K. S. Nathan, and E. J. Bellegarda, "Supervised hidden Markov modeling for on-line handwriting recognition," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 1994.
[2] W. Cho, S-W. Lee, and J. H. Kim, "Modeling and recognition of cursive words with hidden Markov models," *Pattern Recognit.*, vol. 28, no. 12, pp. 1941–1953, Dec. 1995.
[3] M. Gilloux, M. Leroux, and J-M. Bertille, "Strategies for cursive script recognition using hidden Markov models," *Machine Vision Applicat.*, vol. 8, no. 4, pp. 197–205, 1995.
[4] M. Schenkel, I. Guyon, and D. Henderson, "On-line cursive script recognition using time delay neural networks and hidden Markov models," in *Proc. ICASSP, IEE Int. Conf. Acoust., Speech, Signal Process.*, 1994, vol. 2, pp. 637–640.
[5] S. Garcia-Salicetti, P. Gallinari, B. Dorizzi, A. Mellouk, and D. Fanchon, "Neural predictive approach for on-line cursive script recognition," in *Proc. ICASSP, IEE Int. Conf. Acoust., Speech, Signal Process.*, vol. 5, pp. 3463–3466, 1995.
[6] H. Nishida, "Model-based shape matching with structural feature grouping," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 17, Mar. 1995.
[7] M. Parizeau and R. Plamondon, "A fuzzy-syntactic approach to recognition," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 17, July 1995.
[8] Y. Singer and N. Tishby, "Dynamical encoding of cursive handwriting," *Biolog. Cybern.*, vol. 71, no. 3, pp. 227–237, 1994.
[9] L. Schomaker, "Using stroke-or character-based self-organizing maps in the recognition of on-line, connected cursive script," *Pattern Recognit.*, vol. 26, no. 3, pp. 443–450, Mar. 1993.
[10] S. Manke, M. Finke, and A. Waibel, "Combining bitmaps with dynamic writing information for on-line handwriting recognition," in *Proc. 12th IAPR Int. Conf. Pattern Recogn.*, vol. 2, pp. 596–598, 1994.
[11] Y. Le Cunn, L. Bottou, and Y. Bengio, "Reading checks with multilayer graph transformer networks," in *Proc. ICASSP*, 1997, to be published.
[12] G. M. Man and J. C. Poon, "Cursive script segmentation and recognition by dynamic programming," in *Proc. SPIE, Int. Soc. Opt. Eng.*, 1993, vol. 1906, pp. 184–194.
[13] K. Fukushima and T. Imagawa, "Recognition and segmentation of connected characters with selective attention," *Neural Networks*, vol. 6, no. 1, pp. 33–41, 1993.
[14] M. Kadirkamanathan and P. J. W. Rayner, "Unified approach to on-line cursive script segmentation and feature extraction," in *Proc. ICASSP, IEEE Int. Conf. Acoust., Speech, Signal Process.*, vol. 3, pp. 1659–1662, 1993.
[15] G. De Micheli, *Synthesis and Optimization of Digital Circuits*. New York: McGraw-Hill, 1994.
[16] I-J. Lin and S. Y. Kung, "A recursively structured solution for hand writing and speech recognition," in *Proc. IEEE Workshop Multimedia Signal Process.*, 1997, pp. 587–592.
[17] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. Cambridge, MA: MIT Press, 1990.
[18] S. Y. Kung, *VLSI Array Processors*. Englewood Cliffs, NJ: Prentice-Hall, 1993.
[19] ———, *Digital Neural Networks*. Englewood Cliffs, NJ: Prentice-Hall, 1993.
[20] (http://www.ee.princeton.edu/~ijonglin/yapr/READ_ME.html)
[21] Entry on cryptography, *Encyclopedia Britannica*.
[22] A. W. Senior, "Off-line handwriting recognition: A review and experiments," Tech. Rep. CUED/F-INFENG/TR 105, Eng. Dept., Cambridge Univ., Cambridge, U.K., Dec. 1992.



I-Jong Lin received the B.S. degree in computer system engineering and the M.S. degree in electrical engineering from Stanford University, Stanford, CA, in 1993 and 1995, respectively. He received the M.A. degree from Princeton University, Princeton, NJ, in 1997, where he is currently pursuing the Ph.D. degree in electrical engineering.



Sun-Yuan Kung (F'88) received the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA.

In 1974 he was an Associate Engineer with Amdahl Corporation, Sunnyvale, CA. From 1977 to 1987, he was a Professor of Electrical Engineering Systems, University of Southern California, Los Angeles. Since 1987, he has been a Professor of Electrical Engineering, Princeton University, Princeton, NJ. Since 1990, he has served as an Editor-in-Chief of the *Journal of VLSI Signal Processing*. Recently, he served as a General Chair of the *1997 IEEE Workshops on Multimedia Signal Processing* (Princeton University).

Dr. Kung received the 1992 IEEE Signal Processing Society's Technical Achievement Award for his contributions on "parallel processing and neural network algorithms for signal processing." He was appointed an IEEE SP Distinguished Lecturer in 1994. He received the 1996 IEEE Signal Processing Society's Best Paper Award. He has authored more than 300 technical publications including three books: *VLSI Array Processors* (Englewood Cliffs, NJ: Prentice-Hall, 1988) (with Russian and Chinese translations), *Digital Neural Networks* (Englewood Cliffs, NJ: Prentice Hall, 1993), and *Principal Component Neural Networks* (New York: Wiley, 1996).