Return to Main

Objectives

Overview:
Technology
Optimal Search

Description:
Optimality
Example
Variants
Discussion

On-Line Resources:
Trick: Tutorial
Cassidy: Dynamic Time-Warping
DTW Applet

LECTURE 21: DYNAMIC PROGRAMMING

- Objectives:
  - Historical significance of dynamic programming
  - Introduce a fixed-endpoint solution
  - Understand optimality constraints
  - Explain use in speech recognition

This material can be found in most older speech recognition textbooks. This is the book we recommend:

> J. Deller, et. al., *Discrete-Time Processing of Speech Signals*, MacMillan Publishing Co., ISBN: 0-7803-5386-2, 2000.

For a more detailed description of the use of this technology in speech recognition, see:

- H.F. Silverman, D.P. Morgan, "The Application of Dynamic Programming to Connected Speech Recognition," *IEEE Acoustics, Speech, and Signal Processing Magzine*, vol. 7, pp. 6-25, July 1990.

ECE 8463: FUNDAMENTALS OF SPEECH RECOGNITION

Professor Joseph Picone
Department of Electrical and Computer Engineering
Mississippi State University

email: picone@isip.msstate.edu
phone/fax: 601-325-3149; office: 413 Simrall
URL: http://www.isip.msstate.edu/resources/courses/ece_8463

Modern speech understanding systems merge interdisciplinary technologies from Signal Processing, Pattern Recognition, Natural Language, and Linguistics into a unified statistical framework. These systems, which have applications in a wide range of signal processing problems, represent a revolution in Digital Signal Processing (DSP). Once a field dominated by vector-oriented processors and linear algebra-based mathematics, the current generation of DSP-based systems rely on sophisticated statistical models implemented using a complex software paradigm. Such systems are now capable of understanding continuous speech input for vocabularies of hundreds of thousands of words in operational environments.

In this course, we will explore the core components of modern statistically-based speech recognition systems. We will view speech recognition problem in terms of three tasks: signal modeling, network searching, and language understanding. We will conclude our discussion with an overview of state-of-the-art systems, and a review of available resources to support further research and technology development.

Tar files containing a compilation of all the notes are available. However, these files are large and will require a substantial amount of time to download. A tar file of the html version of the notes is available here. These were generated using wget:

```
wget -np -k -m http://www.isip.msstate.edu/publications/courses/ece_8463/lectures/current
```

A pdf file containing the entire set of lecture notes is available here. These were generated using Adobe Acrobat.

Questions or comments about the material presented here can be directed to help@isip.msstate.edu.

LECTURE 21: DYNAMIC PROGRAMMING

- Objectives:
  - Historical significance of dynamic programming
  - Introduce a fixed-endpoint solution
  - Understand optimality constraints
  - Explain use in speech recognition

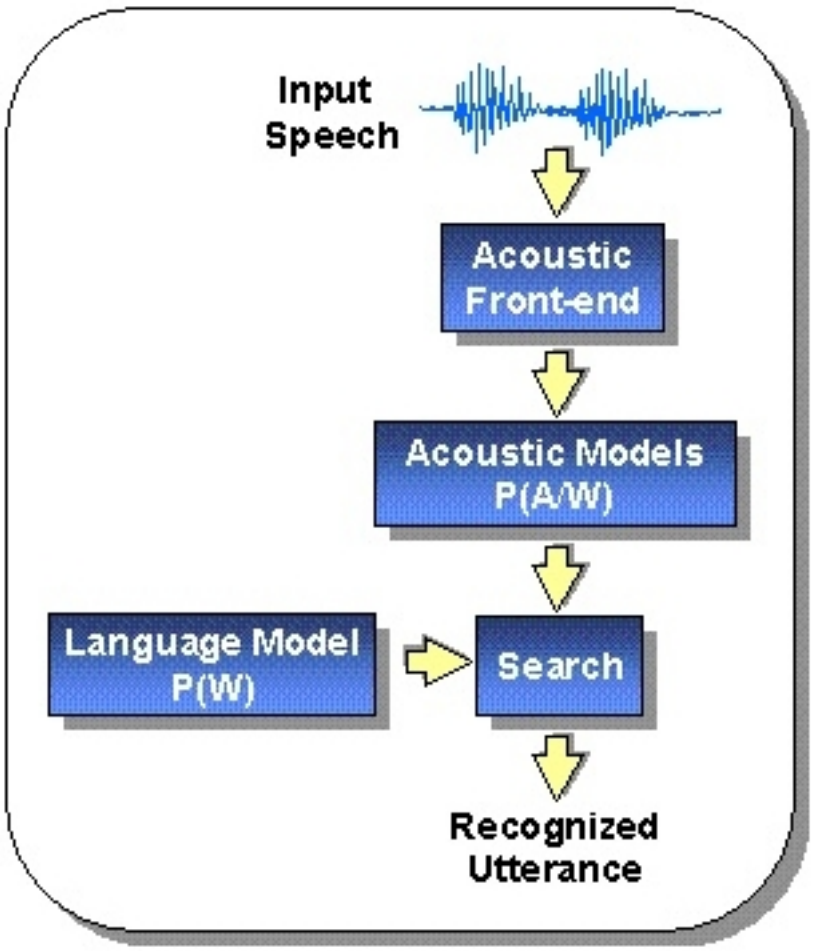
This material can be found in most older speech recognition textbooks. This is the book we recommend:

J. Deller, et. al., *Discrete-Time Processing of Speech Signals*, MacMillan Publishing Co., ISBN: 0-7803-5386-2, 2000.

For a more detailed description of the use of this technology in speech recognition, see:

- H.F. Silverman, D.P. Morgan, "The Application of Dynamic Programming to Connected Speech Recognition," *IEEE Acoustics, Speech, and Signal Processing Magzine*, vol. 7, pp. 6-25, July 1990.

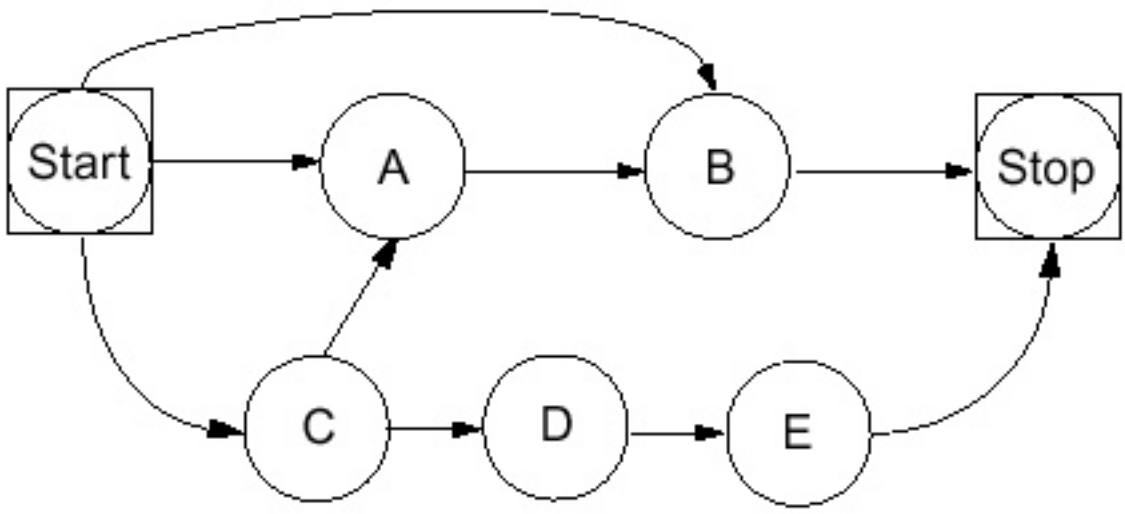
HUMAN LANGUAGE TECHNOLOGY:
SPEECH RECOGNITION IS MULTIDISCIPLINARY



- Acoustic Front-End: Signal Processing
- Acoustic Models: Pattern Recognition, Linguistics
- Language Model: Natural Language Processing
- Search: Computational Linguistics, Cognitive Science

DYNAMIC PROGRAMMING: FAST BUT OPTIMAL SEARCH

- The search space for vocabularies of hundreds of words can become unmanageable if we allow any word to follow any other word (often called the no-grammar case).
- Our rudimentary knowledge of language tells us that, in reality, only a small subset of the vocabulary can follow a given word hypothesis, but that this subset is sensitive to the given word (we often refer to this as "context-sensitive").
- In real applications, user-interface design is crucial (much like the problem of designing GUI's), and normally results in a specification of a language or collection of sentence patterns that are permissible.
- A simple way to express and manipulate this information in a dynamic programming framework is a via a state machine:



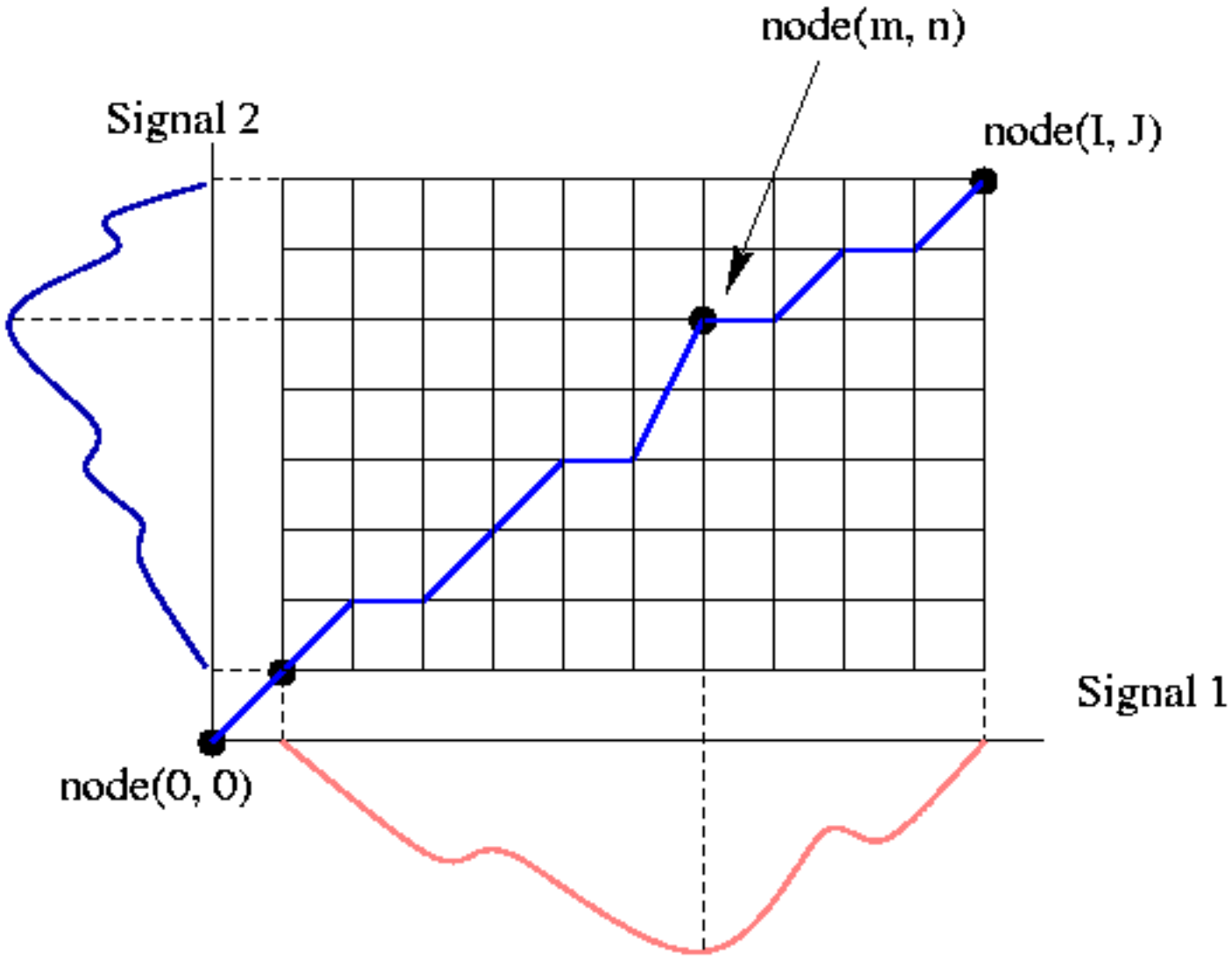
- For example, when you enter state C, you output one of the following words: {daddy, mommy}.

  If:
  state A: give
  state B: me
  state C: {daddy, mommy}
  state D: come
  state E: here

  With such a state machine, we can generate phrases such as:

  Daddy give me
  transitions: Start -> C -> A -> B -> Stop

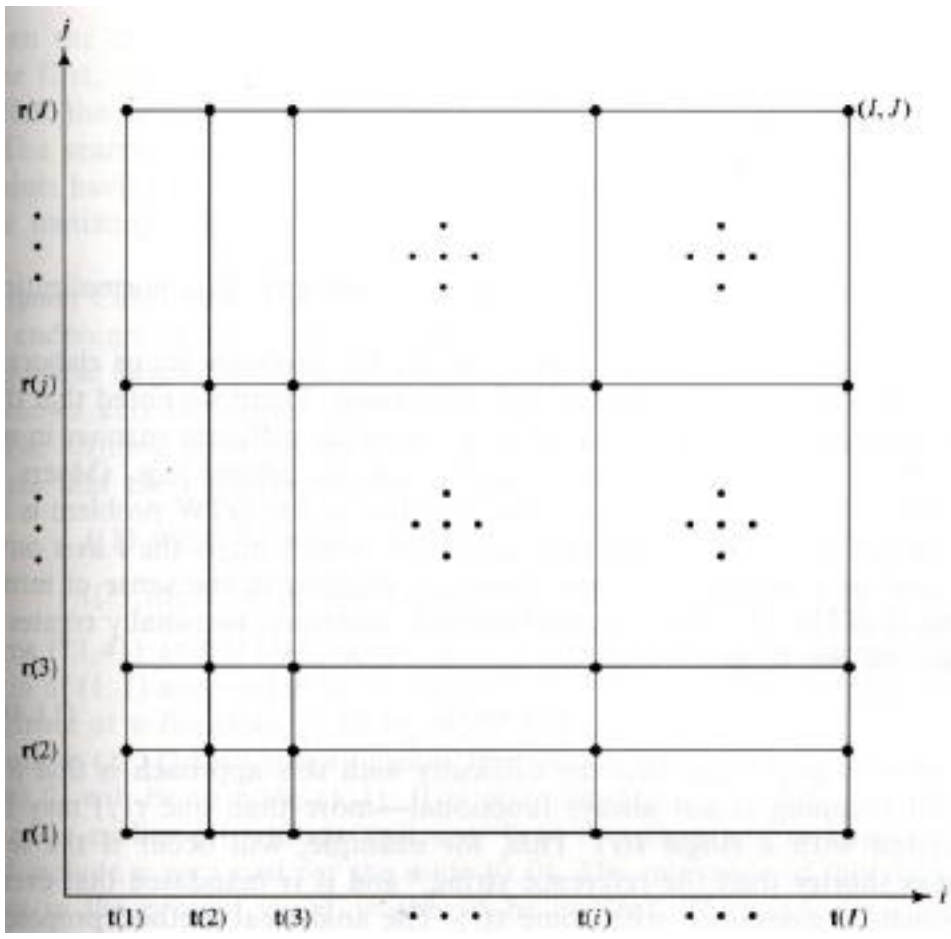
- We can also use this technology to compare feature streams from two signals:



- The latter example is similar to an early approach to speech recognition denoted **dynamic time-warping**. Why might this be useful for speech recognition?

BELLMAN'S PRINCIPLE OF OPTIMALITY

- Consider the discrete space (grid) shown below:



- Define a path from node (s,t) to (u,v) as an n-tuple:

$$(s,t), (i_1,j_1), (i_2,j_2), ..., (u,v)$$

- Define a distance in moving from node i_{k-1},j_{k-1} to i_k,j_k as:

$$d[(i_k,j_k) \mid (i_{k-1},j_{k-1})] = d_T[(i_k,j_k) \mid (i_{k-1},j_{k-1})] + d_N((i_k,j_k)$$

- Define an overall path cost as:

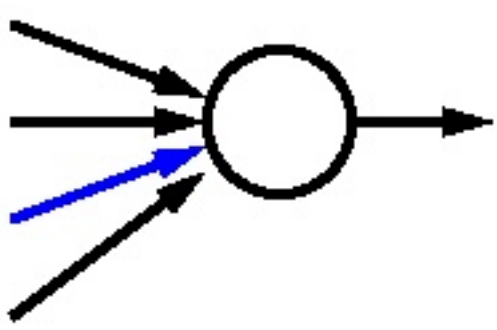
$$D(i,j) = \sum_{k=1}^{K} d[(i_k,j_k) \mid (i_{k-1},j_{k-1})]$$

- **Bellman's Principle of Optimality**:

$$(s,t) \rightarrow (u,v) = (s,t) \rightarrow (w,x) \oplus (w,x) \rightarrow (u,v)$$

for any $s, t, u, v, w,$ and x such that $0 \le s, w, u \le I$ and $0 \le t, x, v \le J$, where $\oplus$ denotes concatenation of path segments.

- This theorem has remarkable consequences: We do not need to exhaustively search for the best path. Instead, we can build the best path by considering a sequence of partial paths, and retaining the best local path:

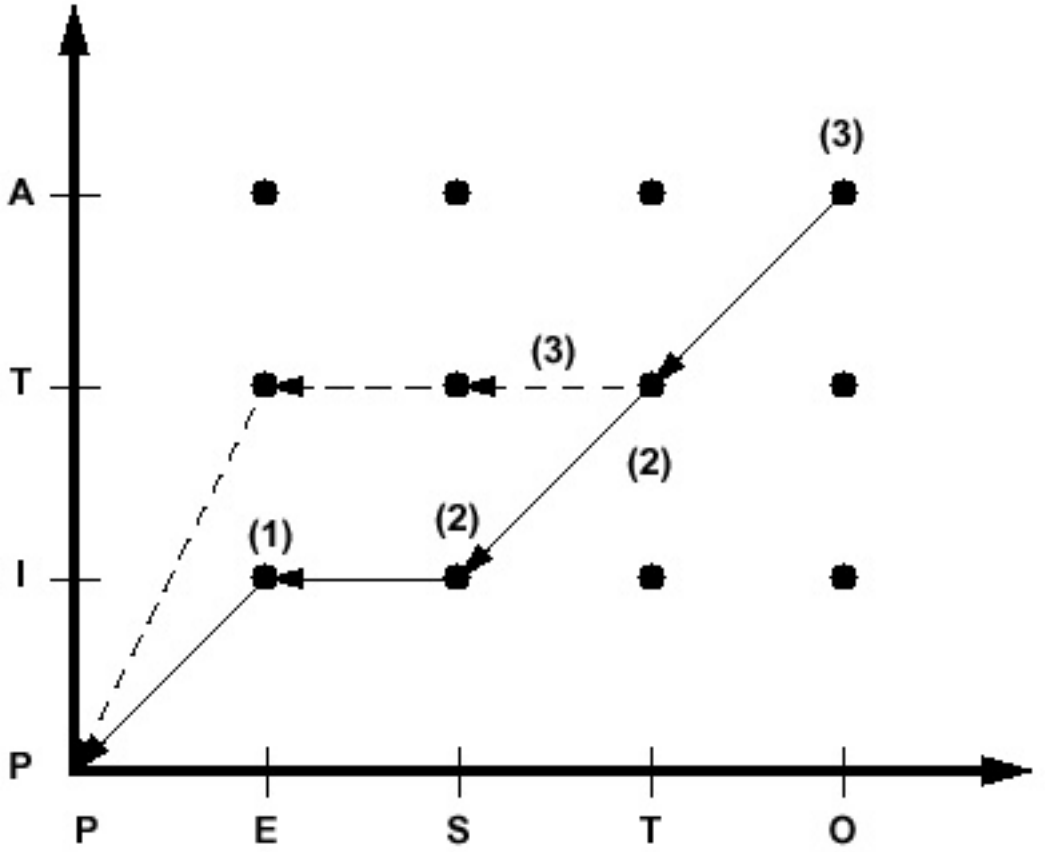


- The savings in computations are enormous: O(KVL) vs. $O(KV^L)$.

- For this reason, dynamic programming is one of the most widely used algorithms in computing. It has been applied to many areas of speech recognition including language modeling (search) and scoring (string edits).

EXAMPLE: STRING SIMILARITY

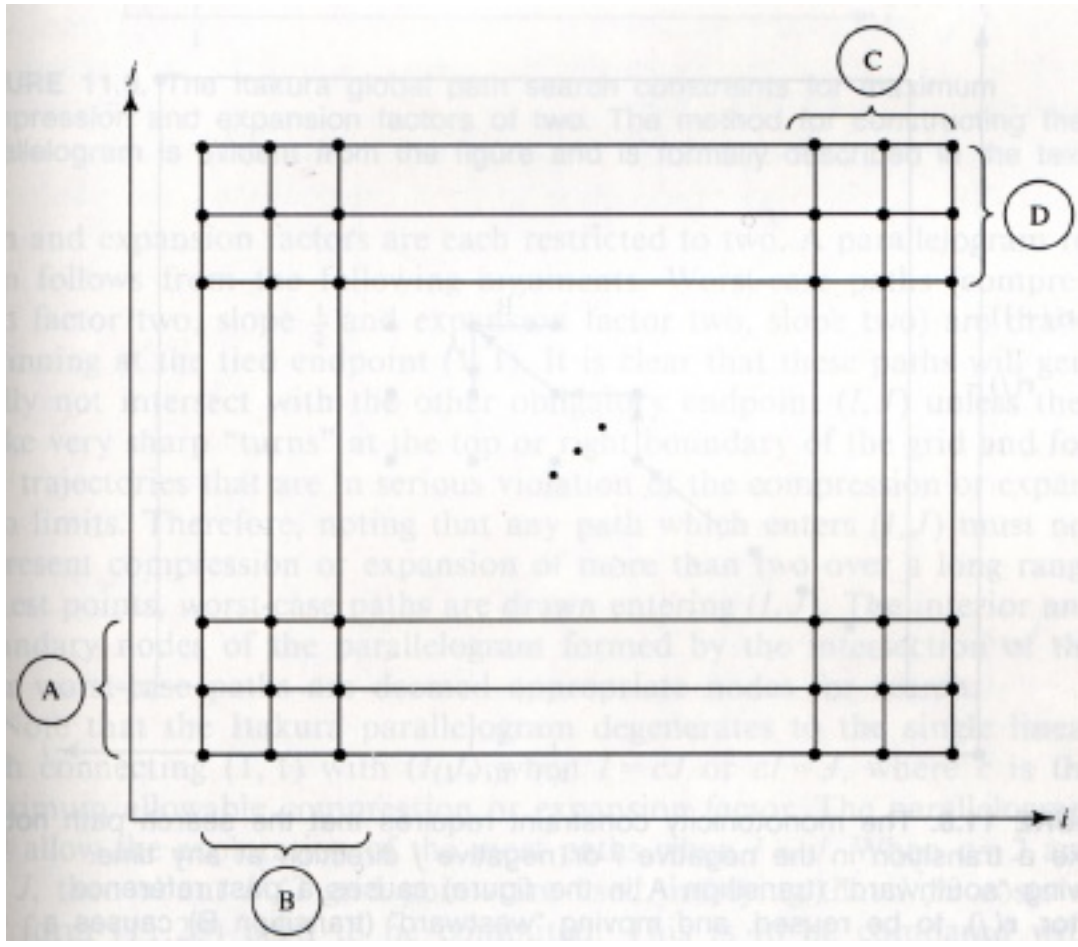
- Consider the problem of measuring the distance between two strings below:



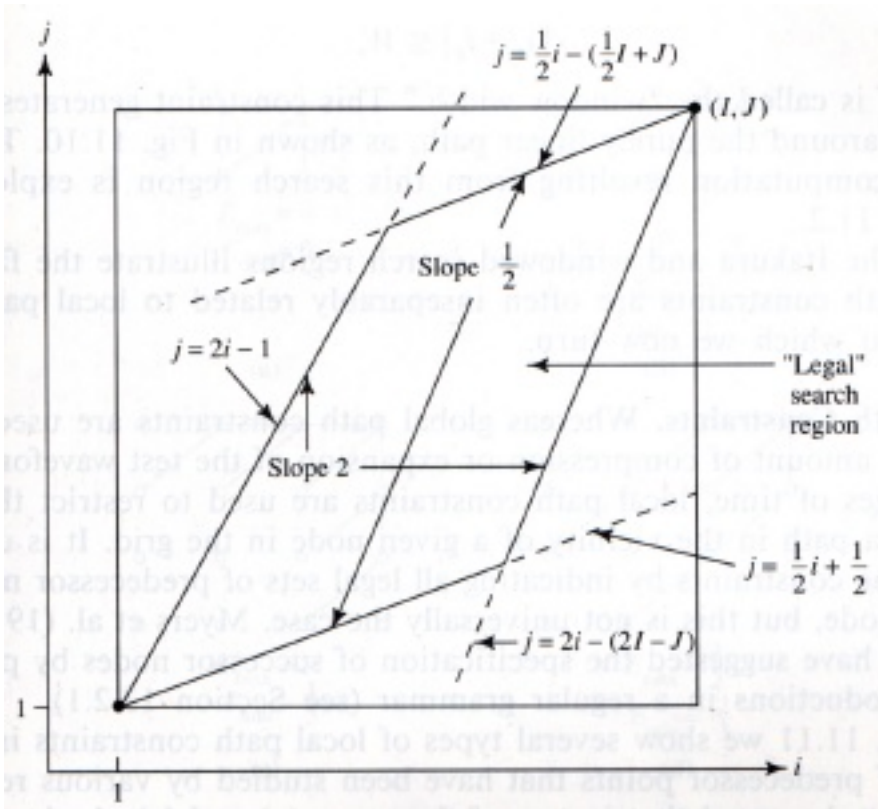
- **Transition Penalty**: Any non-diagonal transition has a penalty of 1 unit.
- **Node Penalty**: Any two dissimilar letters that are matched at a node incur a penalty of 1.
- How much memory do we need if we just want to keep the best score?
- When would we want to retain the backpointers at each node?
- How can we constrain the search space to make this algorithm more efficient for speech processing?

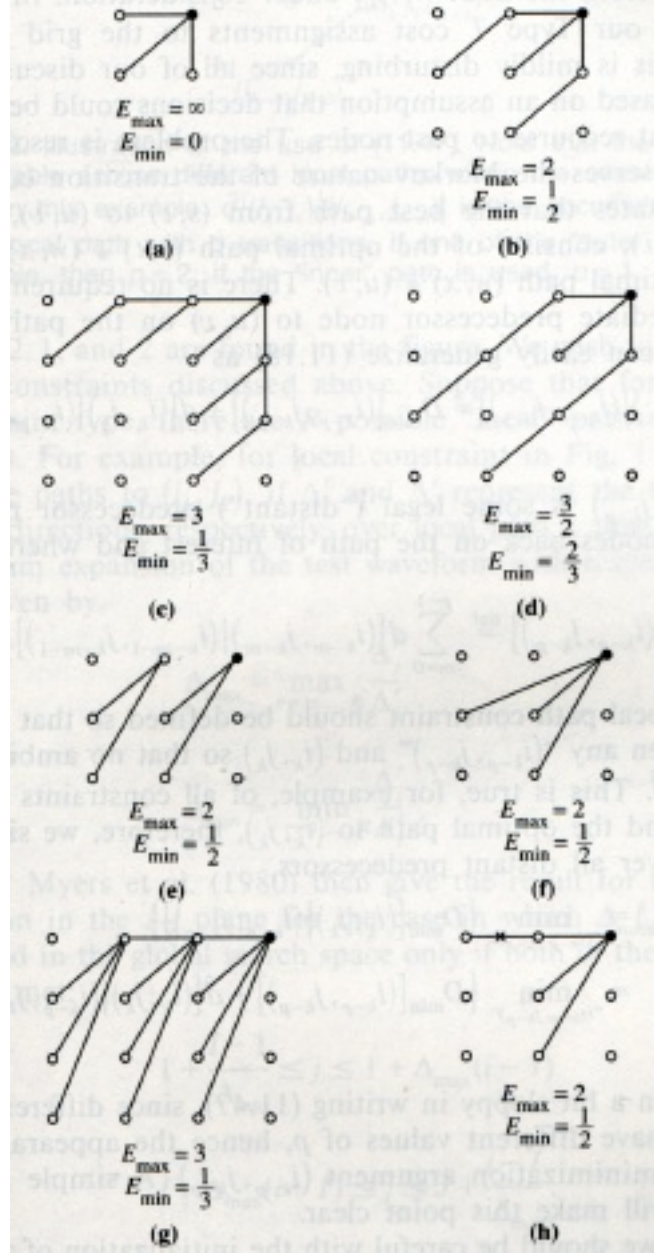
COMMON VARIANTS OF THE DYNAMIC PROGRAMMING FRAMEWORK

- Previously, we demonstrated **fixed-endpoint** DP.
- Relaxed endpoint and free endpoint solutions are also possible:



- Since the speech signal (or text) flows left to right, we can impose "slope constraints" to improve performance and decrease computation:





- The slope constraints are implemented by limiting the search space and imposing transition penalties.
- What aspects of the speech signal influence the design of the slope constraints?

DISCUSSION: HOW DO WE APPLY DYNAMIC PROGRAMMING TO SPEECH RECOGNITION?

- The optimal solution is only as good as the cost function. What is an appropriate cost function for speech recognition? What features should we use?
- How should we train our reference models ("templates")?
- How can we attempt continuous speech recognition using this approach?
  (Hints: Word Spotting, Level Building, Bridle Algorithm)

A Tutorial on Dynamic Programming

Michael A. Trick

Mini V, 1997

Michael A. Trick
Sun Jun 14 13:05:46 EDT 1998

Dynamic Time Warping

One of the earliest approaches to isolated word speech recognition was to store a prototypical version of each word (called a template) in the vocabulary and compare incoming speech with each word, taking the closest match. This presents two problems: what form do the templates take and how are they compared to incoming signals.

The simplest form for a template is a sequence of feature vectors -- that is the same form as the incoming speech. We will assume this kind of template for the remainder of this discussion. The template is a single utterance of the word selected to be typical by some process; for example, by choosing the template which best matches a cohort of training utterances.

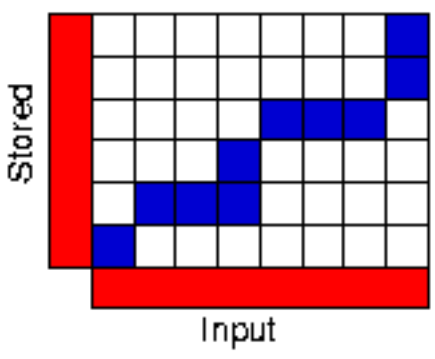
Comparing the template with incoming speech might be achieved via a pairwise comparison of the feature vectors in each. The total distance between the sequences would be the sum or the mean of the individual distances between feature vectors. The problem with this approach is that if a constant window spacing is used, the lengths of the input and stored sequences is unlikely to be the same. Moreover, within a word, there will be variation in the length of individual phonemes: *Cassidy* might be uttered with a long /A/ and short final /i/ or with a short /A/ and long /i/. The matching process needs to compensate for length differences and take account of the non-linear nature of the length differences within the words.

The Dynamic Time Warping algorithm achieves this goal; it finds an optimal match between two sequences of feature vectors which allows for streached and compressed sections of the sequence. The paper by Sakoe and Chiba (Dynamic Programming Algorithm Optimisation for Spoken Word Recognition) gives a detailed description of the algorithm; I will summarise it here.

The DTW Grid

We can arrange the two sequences of observations on the sides of a grid (Figure 6-1>) with the unknown sequence on the bottom (six observations in the example) and the stored template up the left hand side (eight observations). Both sequences start on the bottom left of the grid. Inside each cell we can place a distance measure comparing the corresponding elements of the two sequences.

Figure 6-1. An example DTW grid



To find the best match between these two sequences we can find a path through the grid which minimises the total distance between them. The path shown in blue in Figure 6-1> gives an example. Here, the first and second elements of each sequence match together while the third element of the input also matches best against the second element of the stored pattern. This corresponds to a section of the stored pattern being stretched in the input. Similarly, the fourth element of the input matches both the second and third elements of the stored sequence: here a section of the stored sequence has been compressed in the input sequence. Once an overall best path has been found the total distance between the two sequences can be calculated for this stored template.

The procedure for computing this overall distance measure is to find all possible routes through the grid and for each one of these compute the overall distance. The overall distance is given in Sakoe and Chiba, Equation 5, as the minimum of the sum of the distances between individual elements on the path divided by the sum of the warping function (which will be discussed later). The division is to make paths of different lengths comparable.

It should be apparent that for any reasonably sized sequences, the number of possible paths through the grid will be very large. In addition, many of the distance measures could be avoided since the first element of the input is unlikely to match the last element of the template for example. The DTW algorithm is designed to exploit some observations about the likely solution to make the comparison between sequences more efficient.

Optimisations

The major optimisations to the DTW algorithm arise from observations on the nature of good paths through the grid. These are outlined in Sakoe and Chiba and can be summarised as:

- *Monotonic condition*: the path will not turn back on itself, both the i and j indexes either stay the same or increase, they never decrease.
- *Continuity condition*: The path advances one step at a time. Both i and j can only increase by 1 on each step along the path.
- *Boundary condition*: the path starts at the bottom left and ends at the top right.
- *Adjustment window condition*: a good path is unlikely to wander very far from the diagonal. The distance that the path is allowed to wander is the window length *r*.
- *Slope constraint condition*: The path should not be too steep or too shallow. This prevents very short sequences matching very long ones. The condition is expressed as a ratio n/m where m is the number of steps in the x direction and m is the number in the y direction. After m steps in x you must make a step in y and vice versa.

By applying these observations we can restrict the moves that can be made from any point in the path and so restrict the number of paths that need to be considered. For example, with a slope constraint of P=1, if a path has already moved one square up it must next move either diagonally or to the right.

The power of the DTW algorithm goes beyond these observations though. Instead of finding all possible routes through the grid which satisfy these constraints, the DTW algorithm works by keeping track of the cost of the best path to each point in the grid. During the match process we have no idea which path is the lowest cost path; but this can be traced back when we reach the end point.

The Weighting Function

A path through the grid is written in the paper as *F=c(1),c(2)...c(K)*, the generalised element of the path is *c(k)* and this consists of a pair of coordinates in the *i* (input) and *j* (stored) directions. The i coordinate of the kth path element is *i(k)*.

The weighting function *w(k)* introduced into the overall distance measure is used to normalise for the path length. Two alternate weighting functions are presented: symmetric and asymmetric. Both functions are derived from the distance travelled (in grid units) in the last step of the path. The symmetric form combines the i and j directions while the asymmetric form uses just the i direction.

If the path has just made a diagonal step then i and j both increase by 1 and the symmetric w(k) = 1+1 = 2; the asymmetric w(k) = 1; The sum of this function over the length of the path gives a measure of how long the path is. This is used in normalising the overall distance measures.

DYNAMIC TIME WARPING DIGIT RECOGNIZER

"All applets on our web site now require the java plugin to run. This is necessary so we can bring state-of-the-art features to you which are not currently supported by browser vendors. You can find the plugin at http://java.sun.com/products/plugin. For additional information or suggestions please contact help@isip.msstate.edu"

- Tutorial: Learn how to use this applet.
- View Source Code: View the source code for this applet.

All applets on our web site now require a Java plug-in. This is necessary so we can bring state-of-the-art Java features to you which are not currently supported by browser vendors such as Netscape. You can find the appropriate plug-in at http://java.sun.com/products/plugin. We have generated a list of steps necessary for installing the plug-in in a Unix environment. For additional information or help with your installation please contact help@isip.msstate.edu.
