(73)

Application of stochastic grammars to understanding action

by

Yuri A. Ivanov

M.S., Computer Science
State Academy of Air and Space Instrumentation, St. Petersburg, Russia
February 1992

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE IN MEDIA ARTS AND SCIENCES
at the
Massachusetts Institute of Technology
February 1998

Signature of Author _____
Program in Media Arts and Sciences
January 23, 1997

Certified by _____
Aaron F. Bobick
LG Electronics Inc., Career Development
Assistant Professor of Computational Vision
Program in Media Arts and Sciences
Thesis Supervisor

Accepted by _____
Stephen A. Benton
Chairperson
Departmental Committee on Graduate Students
Program in Media Arts and Sciences

Application of stochastic grammars to understanding action

by
Yuri A. Ivanov

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning
on January 23, 1997
in partial fulfillment of the requirements for the degree of

Master of Science in Media Arts and Sciences

Abstract

This thesis addresses the problem of using probabilistic formal languages to describe and understand actions with explicit structure. The thesis explores several mechanisms of parsing the uncertain input string aided by a stochastic context-free grammar. This method, originating in speech recognition, lets us combine a statistical recognition approach with a syntactical one in a unified syntactic-semantic framework for action recognition.

The basic approach is to design the recognition system in a two-level architecture. The first level, a set of independently trained component event detectors, produces the likelihoods of each component model. The outputs of these detectors provide the input stream for a stochastic context-free parsing mechanism. Any decisions about supposed structure of the input are deferred to the parser, which attempts to combine the maximum amount of the candidate events into a most likely sequence according to a given Stochastic Context-Free Grammar (SCFG). The grammar and parser enforce longer range temporal constraints, disambiguate or correct uncertain or mis-labeled low level detections, and allow the inclusion of a priori knowledge about the structure of temporal events in a given domain.

The method takes into consideration the continuous character of the input and performs "structural rectification" of it in order to account for misalignments and ungrammatical symbols in the stream. The presented technique of such a rectification uses the structure probability maximization to drive the segmentation.

We describe a method of achieving this goal to increase recognition rates of complex sequential actions presenting the task of segmenting and annotating input stream in various sensor modalities.

Thesis Supervisor: Aaron F. Bobick
Title: Assistant Professor of Computational Vision

Application of stochastic grammars to understanding action

by
Yuri A. Ivanov

The following people served as readers for this thesis:

Reader: _____

Alex Pentland
Toshiba Professor of Media Arts and Sciences
Media Laboratory, Massachusetts Institute of Technology

Reader: _____

Walter Bender
Associate Director for Information Technology
Media Laboratory, Massachusetts Institute of Technology

Acknowledgments

First and foremost I would like to thank my advisor, professor Aaron Bobick, for giving me the opportunity to join his team at MIT and introducing me to machine vision. I really appreciated his open-mindedness, readiness to help and listen and giving me the freedom to see things my own way. Every discussion with him taught me something new. I consider myself very fortunate to have him as my advisor - both personally and professionally he has given me more than I could ever ask for. Thanks, Aaron.

I would like to thank my readers, Walter Bender and Sandy Pentland, for taking the time from their busy schedules to review the thesis and for their comments and suggestions.

This work would not be possible without the input from all members of the High Level Vision group (in no particular order - Drew, Stephen, Lee, Claudio, Jim, John, Chris and Martin). My special thanks to Andrew Wilson, who had the misfortune of being my officemate for the last year and a half and who had the patience to listen to all my ramblings and answer all my endless questions to the degree I never expected.

But I guess this whole thing wouldn't even have happen if George "Slowhand" Thomas hadn't come along and told me that while I am looking for better times, they pass. Thanks, bro.

Thanks to Mom and Dad for giving me all their love and always being there for me, regardless what mood I was in.

Thanks to Margie for going the extra ten miles every time I needed it.

And, finally, thanks to my wife of "has it been that long already?" years, Sarah. Thank you for being so supportive, loving, patient and understanding. Thank you for putting up with all my crazy dreams and making them all worth trying. For you, I only have one question: "You know what?"

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Motivation

In the last several years there has been a tremendous growth in the amount of computer vision research aimed at understanding *action.* As noted by Bobick [5] these efforts have ranged from the interpretation of basic movements such as recognizing someone walking or sitting, to the more abstract task of providing a Newtonian physics description of the motion of several objects.

In particular, there has been emphasis on activities or behaviors where the entity to be recognized may be considered as a stochastically predictable sequence of states. The greatest number of examples come form work in gesture recognition [30, 6, 28] where analogies to speech and handwriting recognition inspired researchers to devise *hidden Markov model* methods for the classification of gestures. The basic premise of the approach is that the visual phenomena observed can be considered Markovian in some feature space, and that sufficient training data exists to automatically learn a suitable model to characterize the data.

Hidden Markov Models ([22]) have become the weapon of choice of the computer vision community in the areas where complex sequential signals have to be learned and recognized. The popularity of HMMs is due to the richness of the mathematical foundation of the model, its robustness and the fact that many important problems are easily solved by application of this method.

While HMMs are well suited for modeling parameters of a stochastic process with assumed Markovian properties, their capabilities are limited when it comes to capturing and expressing the structure of a process.

If we have a reason to believe that the process has primitive components, it might be beneficial to separate the interrelations of the process components from their internal statistical properties in the representation of the process. Such a separation lets us avoid over-generalization and employ the methods which are specific to the analysis of either the structure of the signal or its content.

1.1 Motivation

Here we establish the motivation for our work based on four lines of argument. First we show how the separation of structural characteristics from statistical parameters in a vision task can be useful. Then we proceed to discuss the proposed approach in Bayesian formulation, in AI framework and, finally, present the motivation in context of recognition of a conscious directed task.

1.1.1 Structure and Content

Our research interests lie in the area of vision where observations span extended periods of time. We often find ourselves in a situation where it is difficult to formulate and learn parameters of a model in clear statistical terms, which prevents us from using purely statistical approaches to recognition. These situations can be characterized by one or more of the following properties:

- complete data sets are not always available, but component examples are easily found;
- semantically equivalent processes possess radically different statistical properties;
- competing hypotheses can absorb different lengths of the input stream raising the need for naturally supported temporal segmentation;
- structure of the process is difficult to learn but is explicit and a priori known;
- the process' structure is too complex, and the limited Finite State model which is simple to estimate, is not always adequate for modeling the process' structure.

Many applications exist where purely statistical representational capabilities are limited and surpassed by the capabilities of the structural approach.

Structural methods are based on the fact that often the significant information in a pattern is not merely in the presence, absence or numerical value of some feature set. Rather, the interrelations or interconnections of features yield most important information, which facilitates structural description and classification. One area, which is of particular interest is the domain of syntactic pattern recognition. Typically, the syntactic approach formulates hierarchical descriptions of complex patterns built up from simpler sub-patterns. At the lowest level, the primitives of the pattern are extracted from the input data. The choice of the primitives distinguishes the syntactical approach from any other. The main difference is that the features are just any available measurement which characterizes the input data. The primitives, on the other hand, are sub-patterns or building blocks of the structure.

Superior results can be achieved by combining statistical and syntactical approaches into a unified framework in a statistical-syntactic approach. These methods have already gained popularity in AI and speech processing (Eg. [17]).

The statistical-syntactic approach is applicable when there exists a natural separation of structure and contents. For many domains such a division is clear. For example, consider ballroom dancing. There are a small number of primitives (e.g. `right-leg-back`) which are then structured into higher level units (e.g. `box-step`, `quarter-turn`, etc.). Typically one will have many examples of `right-leg-back` drawn from the relatively few examples each of the higher level behaviors. Another example might be recognizing a car executing a parallel parking maneuver. The higher level activity can be described as first a car executes an `pull-along-side` primitive followed by an arbitrary number of cycles through the pattern `turn-wheels-left`, `back-up`, `turn-wheels-right`, `pull-forward`. In these instances, there is a natural division between atomic, statistically abundant primitives and higher level coordinated behavior.

1.1.2 Bayesian View

Suppose we have a sequence of primitives composing the action structure S and the observation sequence O. We assume a simple probabilistic model of an action where particular S produces some O with probability $P(S,O)$. We need to find $\hat{S}$ such that $P(\hat{S}|O) = \max_S P(S|O)$.

Using Bayes rule:

$$P(S|O) = \frac{P(O|S)P(S)}{P(O)} \tag{1.1}$$

The MAP estimate of $\hat{S}$ is, therefore,

$$\hat{S} = arg \max_{S} P(O|S)P(S)$$

The first term, $P(O|S)$, is a component likelihood (termed *acoustic model* in speech recognition), representing probability of the observation given the sequence of primitives. It is obtained via recognition of the primitive units according to some statistical model. The second term, $P(S)$ is a structure model (also called a *language model*) as it describes the probability associated with a postulated sequence of primitives.

Typically, when neither term is known they are assumed to be drawn from a distribution of some general form and their parameters are estimated.

The motivation of this work arises from the need to model action-based processes where the structure model can be arbitrarily complex. While the component likelihood is based on variations in the signal and a statistical model is typically appropriate, the structure model in our context is based on behavior, or directed task. The corresponding probability distribution is often non-trivial and difficult to model by purely statistical means.

De-coupling between statistical models of the structure and the underlying primitives is desirable in our context, since it makes it possible to deploy the appropriate machinery at each level. In order to do that we need to develop a mechanism for independent estimation of models at each level and propagation of constraints through such a two-level structure.

There have been attempts to de-couple structural knowledge from component knowledge and express the structure via establishing interrelations of the components, realizing them in the architecture of the recognition system. For instance, in speech recognition, grammar networks are widely used to impose syntactic constraints on the recognition process. One such example is the parsing strategy employed by an HMM Tool Kit, developed by Entropic Research Lab. Individual temporal feature detectors can be tied together according to the expected syntax. The syntactic model is expressed by a regular grammar in extended Backus-Naur form. This grammar is used to build a network of the detectors and perform long sequence parse by a Viterbi algorithm based on token passing.

The resulting grammar network is shown in figure 1-1a. The structure of the incoming

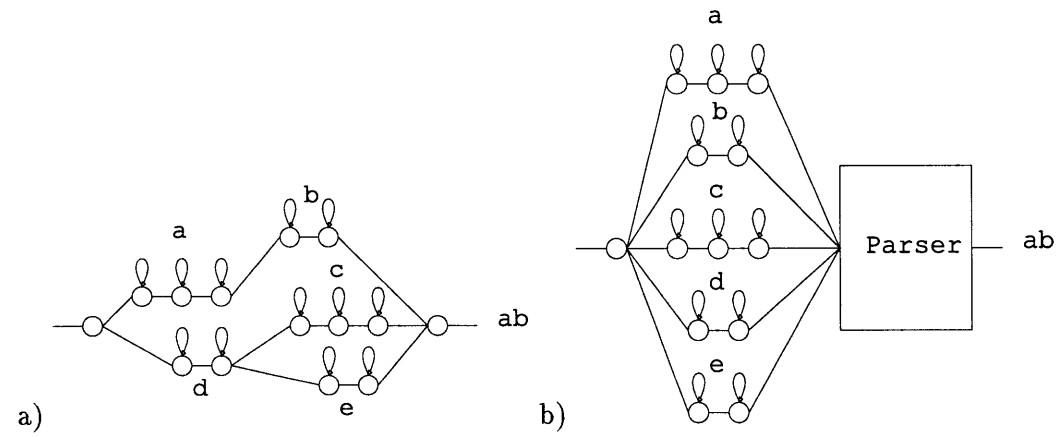


Figure 1-1: Illustration of different parsing strategies. a) Example of HTK – individual temporal feature detectors for symbols *a*, *b*, *c*, *d* and *e* are combined into a grammar network. b) Proposed architecture which achieves de-coupling between the primitive detectors and the structural model (probabilistic parser in this case).

string can be described by a non-recursive, finite state model.

Our approach goes further in de-coupling the structure from content. We do not impose the structural constraints on the detectors themselves. Instead we run them in parallel, deferring the structural decisions to the parser, which embodies our knowledge of the action syntax. A clear advantage of this architecture is that the structural model can take on arbitrary complexity, not limited by an FSM character of the network, according to our beliefs about the nature of the action.

1.1.3 AI View

AI approaches to recognition were traditionally based on the idea of incorporating the knowledge from a variety of knowledge sources and bringing them together to bear on the problem at hand. For instance, the AI approach to segmentation and labeling of the speech signal is to augment the generally used acoustic knowledge with phonemic knowledge, lexical knowledge, syntactic knowledge, semantic knowledge and even pragmatic knowledge.

Rabiner and Juang ([23]) show that in tasks of automatic speech recognition, the word correcting capability of higher-level knowledge sources is extremely powerful. In particular, they illustrate this statement by comparing performance of a recognizer both with and without syntactic constraints (figure[1] 1-2). As the deviation (noise) parameter $SIGMA$ gets larger, the word error probability increases in both cases. In the case without syntactic

[1] Reprinted from [23]

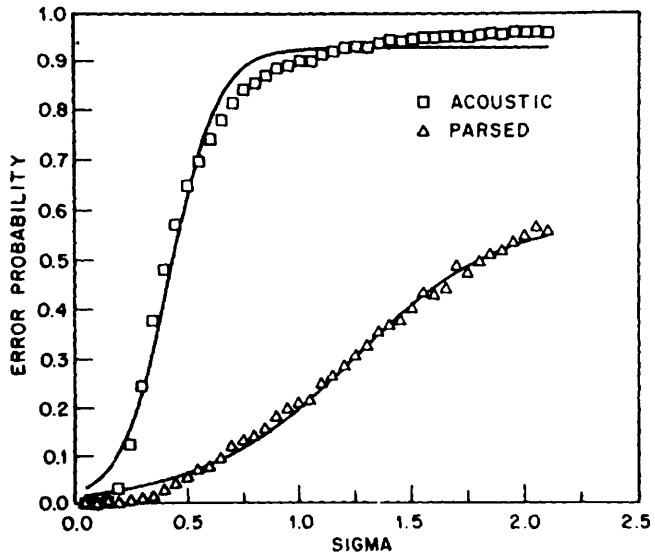


Figure 1-2: Error correcting capabilities of syntactic constraints

constraints, the error probability quickly leads to 1, but with the syntactical constraints enforced, it increases gradually with an increase in noise.

This is the approach to recognition of structured visual information which we take in this work. It allows us to not only relieve the computational strain on the system, but also to make contextual decisions about likelihoods of each of the model as we parse the signal.

1.1.4 Generative Aspect

In problems of visual action recognition we deal with problems which originate in task-based processes. These processes are often perceived as a sequence of simpler semantically meaningful sub-tasks, forming the basis of the task vocabulary.

As we speak of ballet ([11]), we speak of plié and arabesque and how one follows another[2]; as we speak of tennis or volleyball, we speak of backhand and smash and of playing by the rules. This suggests that when we turn to machines for solving the recognition problems in higher level processes which we, humans, are used to dealing with, we can provide the machines with many intuitive constraints, heuristics, semantic labels and rules which are based on our knowledge of the generation of the observation sequence by some entity with

[2] Coincidentally, Webster's dictionary defines ballet as "dancing in which conventional poses and steps are combined with light flowing figures and movements"

predictable behavior.

As we attempt to formulate algorithms for recognition of processes which are easily decomposed into meaningful components, the context, or the structure of it can be simply described rather than learned, and then used to validate the input sequence.

In cognitive vision we can often easily determine the components of the complex behavior–based signal. The assumption is perhaps more valid in the context of conscious directed activity since the nature of the signal is such that it is generated by a task-driven system, and the decomposition of the input to the vision system is as valid as the decomposition of the task, being performed by the observed source, into a set of simpler primitive subtasks (eg. [9, 19, 12]).

1.2 Example I: Explicit Structure and Disambiguation

Consider an example: we will attempt to build a model which would let us perform recognition of a simple hand drawing of a square as the gesture is executed. Most likely, the direction in which "a square" is drawn will depend on whether the test subject is left- or right- handed. Therefore, our model will have to contain the knowledge of both possible versions of such a gesture and indicate that a square is being drawn regardless of the execution style.

This seemingly simple task requires significant effort if using only the statistical pattern recognition techniques. A human observer, on the other hand, can provide a set of useful heuristics for a system which would model the human's higher level perception. As we recognize the need to characterize a signal by these heuristics, we turn our attention to syntactic pattern recognition and combined statistical-syntactic approaches, which would allow us to address the problem stated above.

Having a small vocabulary of prototype gestures `right`, `left`, `up` and `down`, we can formulate a simple grammar G_{square} describing drawing a square:

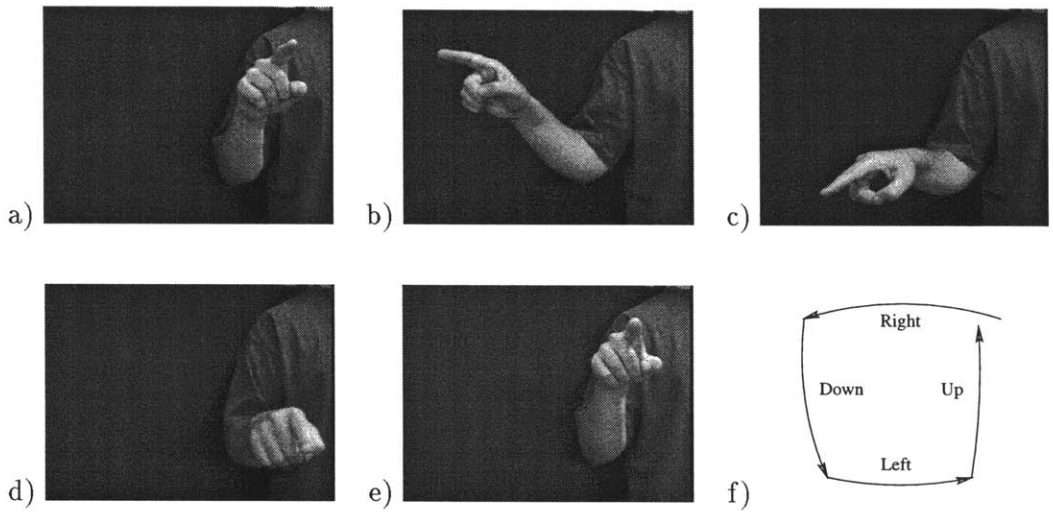


Figure 1-3: Example of the action structure. (a) - (e) Frames 278, 360, 415, 457 and 613 of a sequence showing a square being drawn in the air. (f) shows the motion phases.

Grammar G_{square}:

SQUARE	$\rightarrow$	RIGHT_SQ
	$\rightarrow$	LEFT_SQ
RIGHT_SQ	$\rightarrow$	right down left up
	$\rightarrow$	left up right down
LEFT_SQ	$\rightarrow$	left down right up
	$\rightarrow$	right up left down

We can now identify a simple input string as being "a square" and determine if this particular square is "right handed" or "left handed".

Humans, having observed the video sequence presented in Figure 1-3(a)-(e) have no problem discerning a square. Furthermore, it becomes fairly clear somewhere in the middle of the gesture that the figure drawn will be a square and not something else. This demonstrates the fact that the expected (predicted) structure is based on the "typicality" of the whole gesture and, after the whole gesture has been seen, helps us interpret the arcs in Figure 1-3(f) as straight lines rather than semi-circles.

Now imagine that we are asked to identify the top side of the square. The task has to deal with inherent ambiguity of the process - in our case we cannot really determine if a particular part of the gesture is the top or the bottom because it can be implemented by either `left` or `right` component gestures, depending on which version of the square is being

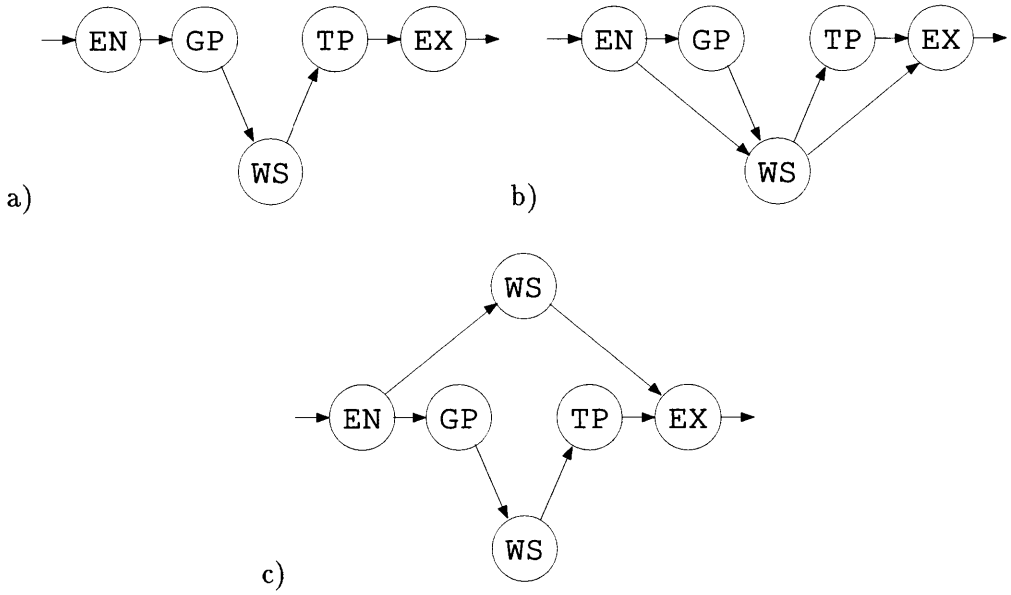


Figure 1-4: Realization of a recursive problem using finite state network. EN – `enter`, GP – `give-pen`, WTS – `wait-till-signed`, TP – `take-pen`, EX – `exit`. a) initial topology, b) added links to take into account the case where boss uses her own pen, c) redundant network to eliminate the invalid paths – EN→GP→WTS→EX and EN→WTS→TP→EX.

drawn. For such locally ambiguous cases disambiguation by context is easily performed in the setting of syntactic recognition. We will revisit this example towards the end of this document.

1.3 Example II: Choosing the Appropriate Model

For the sake of illustration, suppose that an absent-minded secretary keeps losing his pens. Once a day he brings a pile of papers into his boss' office to sign. Quite often he gives the boss his pen to sign the papers and sometimes forgets to take it back. We want to build a system which reminds the secretary that the boss did not return the pen. We can describe the behavior as a sequence of primitive events: `enter`, `give-pen`, `wait-till-signed`, `take-pen` and `exit`. We build a simple sequential network of recognizers which detect the corresponding actions, which is shown in figure 1-4a. Now, if the break in the sequence is detected, we will sound an alarm and remind the secretary to take his pen back.

One complication to the task is that the boss occasionally uses her own pen to sign the documents. At first sight it does not seem to present a problem - we just add two extra arcs to the network (figure 1-4b). But on closer inspection, we realize that the system no

longer works. There is a path through the network which will allow the boss to keep her secretary's pen.

In order to fix the problem we rearrange the network, adding one more recognizer (`wait-till-signed`) and two extra links (figure 1-4c). However, the performance of the original network will slightly drop due to a repeated `wait-till-signed` detector.

A case similar to the "secretary" problem that is of particular importance is the monitoring of assembly-disassembly tasks. The problems of this category possess the property that every action in the beginning of the string has an opposite action in the end. The languages describing such tasks are often referred to as "palindrome" languages. The strings over the "palindrome" language can be described as $L_p : \{\ xy \mid \text{s.t. } y \text{ is a mirror image of } x\}$. It should be clear by now that general palindromes cannot be generated by a finite state machine. However, the bounded instances of such strings can be accepted by an explicit representational enumeration, as was shown above, where the state (`wait-till-signed`) was duplicated. The resulting growth of a finite state machine-based architecture can be prohibitive with increase of the counted nesting.

To substantiate this claim let us look at the relation of the depth of the nesting and the amount of the replicated nodes necessary to implement such a structure. The number of additional nodes (which is proportional to the decrease in performance) required to implement a general recursive system of depth d can be computed by a recurrent relation:

$$\begin{aligned} \mathcal{N}_d &= 2\sum_{i=1}^{d}(d-i+1)N_i \\ N_i &= \begin{pmatrix} d \\ d-i+1 \end{pmatrix} \\ N_0 &= 0 \end{aligned}$$

where $\mathcal{N}_d$ is a total number of nodes in the network. The table 1.1 shows how much larger the network[3] has to be (and, correspondingly, how much slower) to accommodate the "secretary"-type task with increasing value of d.

[3] We assume a simple two-node architecture as a starting point.

d	$\mathcal{N}_d$
2	2
3	16
4	40
5	96
6	224
7	512
8	1,152
9	2,560
10	5,632
...	
20	11,010,048

Table 1.1: Number of additional nodes for a finite state network implementing recursive system of depth d.

1.4 Approach

In this thesis we focus our attention on a mixed statistical-syntactic approach to action recognition. Statistical knowledge of the components is combined with the structural knowledge expressed in a form of grammar. In this framework, the syntactic knowledge of the process serves as a powerful constraint to recognition of individual components as well as recognition of the process as a whole.

To address the issues raised in the previous sections, we design our recognition system in a two-level architecture, as shown in figure 1-1b. The first level, a set of independently trained component event detectors, produces likelihoods of each component model. The outputs of these detectors provide the input stream for a stochastic context-free parsing mechanism. Any decisions about supposed structure of the input are deferred to the parser, which attempts to combine the maximum amount of the candidate events into a most likely sequence according to a given Stochastic Context-Free Grammar (SCFG). The grammar and parser enforce longer range temporal constraints, disambiguate or correct uncertain or mis-labeled low level detections, and allow the inclusion of a priori knowledge about the structure of temporal events in a given domain.

1.5 Outline of the Thesis

The remainder of this thesis proceeds as follows: the next chapter establishes relations of the thesis to previous research in the areas of pattern recognition, motion understanding and

speech processing. Chapter 3 describes the parsing algorithm which is used in this work. Necessary extensions and modifications to the algorithm, which allow the system to deal with uncertain and temporally inconsistent input, are presented in Chapter 4. Chapter 5 gives an overview of the system architecture and its components. Chapter 6 presents some experimental results which were achieved using the method described in this document, and, finally, Chapter 7 concludes the thesis with a brief outline of strong and weak points of the proposed technique.

Chapter 2

Background and Related Work

Inspiration for this work comes primarily from speech recognition, which covers a broad range of techniques. The thesis builds upon research in machine perception of motion and syntactic pattern recognition. This chapter establishes relations of the thesis work to each of the above areas.

2.1 Motion Understanding

In [5] Bobick introduces a taxonomy of approaches to the machine perception of motion. The necessity of such a classification arises from the breadth of the tasks in the domain of motion understanding. The proposed categorization classifies vision methods according to the amount of knowledge which is necessary to solve the problem. The proposed taxonomy makes explicit the representational competencies and manipulations necessary for perception.

According to this classification, the subject of the recognition task is termed *movement*, *activity* or *action*. *Movements* are defined as the most atomic primitives, requiring no contextual or sequence knowledge to be recognized. *Activity* refers to sequences of movements or states, where the only real knowledge required is the statistics of the sequence. And, finally, *actions* are larger scale events which typically include interaction with the environment and causal relationships.

As the objective of the system being developed in this thesis, we define the recognition of complex structured processes. Each of the components of the structure is itself a sequential process, which according to the above taxonomy belongs to the category of *activities*. Ac-

tivity recognition requires statistical knowledge of the corresponding subsequence, modeled in our case by a Hidden Markov Model.

The probabilistic parser performing contextual interpretation of the candidate activities essentially draws upon the knowledge of the domain, which is represented by a grammar. The grammar is formulated in terms of primitive activities (grammar terminals) and their syntactic groupings (grammar non-terminals). Although no explicit semantic inferences are made and no interactions with the environment are modeled in the current implementation of the system, for the scope of this work they were not necessary, implicitly we allow more extensive information to be included in the representation with minimal effort.

In the proposed taxonomy, the complete system covers the range between activities and actions, allowing for cross-level constraint propagation and easy transition from one level to the other.

2.2 Pattern Recognition

The approach to pattern recognition is dictated by the available information which can be utilized in the solution. Based on the available data, approaches to pattern recognition loosely fall into three major categories - *statistical*, *syntactic* or *neural* pattern recognition.

In the instances where there is an underlying and quantifiable statistical basis for the pattern generation *statistical* or *decision-theoretic* pattern recognition is usually employed. Statistical pattern recognition builds upon a set of characteristic measurements, or features, which are extracted from the input data and are used to assign each feature vector to one of several classes.

In other instances, the underlying *structure* of the pattern provides more information useful for recognition. In these cases we speak about *syntactic* pattern recognition. The distinction between syntactic and statistical recognition is based on the nature of the primitives. In statistical approaches, the primitives are *features* which can be any measurements. In syntactic recognition, the primitives must be sub-patterns, from which the pattern, subject to recognition, is composed.

And, finally, *neural* pattern recognition is employed when neither of the above cases hold true, but we are able to develop and train relatively universal architectures to correctly associate input patterns with desired responses. Neural networks are particularly well suited

for pattern association applications, although the distinction between statistical and neural pattern recognition is not as definite and clearly drawn as between syntactical and statistic recognition techniques.

Each of these methodologies offers distinctive advantages and has inevitable limitations. The most important limitations for each of the approaches are summarized in the table below.

Statistical recognition	Syntactic recognition	Neural recognition
Structural information is difficult to express	Structure is difficult to learn	Semantic information is difficult to extract from the network

The technique developed in this thesis is a version of a mixed statistical-syntactic approach. Low level statistics-based detectors utilize the statistical knowledge of the subpatterns of the higher level, syntax-based recognition algorithm, which builds upon our knowledge of the expected structure. This approach allows to integrate the advantages of each of the employed techniques and provides a framework for a cross-level constraint propagation.

2.3 Methods of Speech Recognition

Speech recognition methods span an extremely broad category of problems ranging from signal processing to semantic inference. Most interesting and relevant for our application are the techniques used for determining word and sentence structure.

Most often the N-gram models are used to combine the outputs of word detectors into a meaningful sentence. Given a word sequence W, mutual occurrence of all the words of W is modeled by computing the probabilities:

$$P(W) = P(w_1 w_2 \dots w_l) = P(w_1)P(w_2|w_1)P(w_3|w_1, w_2)\dots P(w_l|w_1 \dots w_{l-1})$$

Such a model essentially amounts to enumeration of all the word sequences and computing their probabilities, which is computationally impossible to do. In practice it is approximated by 2-3 unit long conditional probabilities and the resulting estimation $\hat{P}(W)$ is interpolated if the training data set is limited [23].

Context-free grammars are rarely employed in moderate to large vocabulary natural language tasks. Instead the bi- or tri-gram grammars are used to provide the language model. Perhaps the main reason for that is the fact that CFGs are only a crude approximation of the spoken language structure and CFG statistics are difficult to obtain. N-gram statistics, on the other hand, are relatively straight-forward to compute and to use in practice.

The CFG model assumes a significantly greater role in small to moderate vocabulary tasks, where the domain is more constrained and the task of recognition is to detect relatively short and structured command sequences. In such a setting, the units of recognition are the words of the limited vocabulary[1] and a syntactical description can be formulated with relative ease.

The latter category of speech recognition tasks is where we find the inspiration for our work. Arguably, CFGs are even more attractive in the vision domain since recursive structures in vision are more often the case than in speech.

2.4 Related Work

The proposed approach combines statistical pattern recognition techniques in a unifying framework of syntactic constraints. Statistical pattern recognition in vision has a long history and well developed tools. It is most relevant to the thesis in its application to activity recognition.

One of the earlier attempts to use HMMs for recognition of activities is found in the work by Yamato *et al* ([37]) where discrete HMMs are used to recognize six tennis strokes, performed by three subjects. A 25×25 sub-sampled camera image is used as a feature vector directly.

Activity, as defined above, was the focus of the research where the sequential character of the observation was reflected in the sequential structure of a statistical model, such as work by Darrell ([14]), where the recognition task is performed by a time-warping technique, closely related to HMM methodology.

Examples of statistical representation of sequences are seen in the recent work in understanding human gesture. For instance, Schlenzig *et al* ([28]) describe the results of their

[1] At present time large vocabulary tasks cannot be accomplished on the basis of word primitives due to computational constraints.

experiments of using HMMs for recognition of continuous gestures, which show to be a powerful gesture recognition technique.

Starner and Pentland ([29]) propose an HMM-based approach to recognition of visual language. The task is performed by a set of HMMs trained on several hand signs of American Sign Language (ASL). At run time HMMs output probabilities of the corresponding hand sign phrases. The strings are optionally checked by a regular phrase grammar. In this work the authors fully utilize the advantages of the architecture presented in figure 1-1a.

Wilson, Bobick and Cassell ([35]) analyzed explicit structure of the gesture where the structure was implemented by an equivalent of a finite state machine with no learning involved.

Syntactic pattern recognition is based on the advances of the formal language theory and computational linguistics. The most definitive text on these topics still remains [1]. The authors present a thorough and broad treatment of the parsing concepts and mechanisms.

A vast amount of work in syntactic pattern recognition has been devoted to the areas of image and speech recognition. A review of syntactic pattern recognition and related methods can be found in [27].

Perhaps the most noted disadvantage of the syntactical approach is the fact that the computational complexity of the search and matching in this framework can potentially grow exponentially with the length of the input. Most of the efficient parsing algorithms call for the grammar to be formulated in a certain normal form. This limitation was eliminated for context-free grammars by Earley in the efficient parsing algorithm proposed in his dissertation ([16]). Earley developed a combined top-down/bottom-up approach which is shown to perform at worst at $O(N^3)$ for an arbitrary CFG formulation.

A simple introduction of probabilistic measures into grammars and parsing was shown by Booth and Thompson ([7]), Thomason ([33]), and others.

Aho and Peterson addressed the problem of ill-formedness of the input stream. In [2] they described a modified Earley's parsing algorithm where substitution, insertion and deletion errors are corrected. The basic idea is to augment the original grammar by error productions for insertions, substitutions and deletions, such that any string over the terminal alphabet can be generated by the augmented grammar. Each such production has some cost associated with it. The parsing proceeds in such a manner as to make the total cost minimal. It has been shown that the error correcting Earley parser has the same time and

space complexity as the original version, namely $O(N^3)$ and $O(N^2)$ respectively, where N is the length of the string. Their approach is utilized in this thesis in the framework of uncertain input and multi-valued strings.

Probabilistic aspects of syntactic pattern recognition for speech processing were presented in many publications, for instance in [18, 15]. The latter demonstrates some key approaches to parsing sentences of natural language and shows advantages of use of probabilistic CFGs. The text shows natural progression from HMM-based methods to probabilistic CFGs, demonstrating the techniques of computing the sequence probability characteristics, familiar from HMMs, such as forward and backward probabilities in the chart parsing framework.

An efficient probabilistic version of Earley parsing algorithm was developed by Stolcke in his dissertation ([31]). The author develops techniques of embedding the probability computation and maximization into the Earley algorithm. He also describes grammar structure learning strategies and the rule probability learning technique, justifying usage of Stochastic Context-Free Grammars for natural language processing and learning.

The syntactic approach in Machine Vision has been studied for more than thirty years (Eg. [20, 4]), mostly in the context of pattern recognition in still images. The work by Bunke and Pasche ([10]) is built upon the previously mentioned development by Aho and Peterson ([2]), expanding it to multi-valued input. The resulting method is suitable for recognition of patterns in distorted input data and is shown in applications to waveform and image analysis. The work proceeds entirely in non-probabilistic context.

More recent work by Sanfeliu *et al* ([26]) is centered around two-dimensional grammars and their applications to image analysis. The authors pursue the task of automatic traffic sign detection by a technique based on Pseudo Bidimensional Augmented Regular Expressions (PSB-ARE). AREs are regular expressions augmented with a set of constraints that involve the number of instances in a string of the operands to the star operator, alleviating the limitations of the traditional FSMs and CFGs which cannot count their arguments. More theoretical treatment of the approach is given in [25]. In the latter work, the authors introduce a method of parsing AREs which describe a subclass of a context-sensitive languages, including the ones defining planar shapes with symmetry.

A very important theoretical work, signifying an emerging information theoretic trend in stochastic parsing, is demonstrated by Oomen and Kashyap in [21]. The authors present

a foundational basis for optimal and information theoretic syntactic pattern recognition. They develop a rigorous model for channels which permit arbitrary distributed substitution, deletion and insertion syntactic errors. The scheme is shown to be functionally complete and stochastically consistent.

There are many examples of attempts to enforce syntactic and semantic constraints in recognition of visual data. For instance, Courtney ([13]) uses a structural approach to interpreting action in a surveillance setting. Courtney defines high level discrete events, such as "object appeared", "object disappeared" etc., which are extracted from the visual data. The sequences of the events are matched against a set of heuristically determined sequence templates to make decisions about higher level events in the scene, such as "object A removed from the scene by object B".

The grammatical approach to visual activity recognition was used by Brand ([8]), who used a simple non-probabilistic grammar to recognize sequences of discrete events. In his case, the events are based on blob interactions, such as "objects overlap" etc.. The technique is used to annotate a manipulation video sequence, which has an a priori known structure.

And, finally, hybrid techniques of using combined probabilistic-syntactic approaches to problems of image understanding are shown in [34] and in pioneering research by Fu (eg.[17]).

Chapter 3

Stochastic Parsing

3.1 Grammars and Languages

This section presents brief review of grammar and language hierarchy as given by Chomsky. We discuss the complexity of the model at each level of the hierarchy and show the inherent limitations.

3.1.1 Notation

In our further discussion we will assume the following notation:

Symbol	Meaning	Example
Capital Roman letter	Single non-terminal	A, B, ...
Small Roman letter	Single terminal	a, b, ...
Small Greek letter	String of terminals and non-terminals	λ, μ, ...
Greek letter ϵ	Empty string	ϵ

A minor exception to the above notation are three capital Roman literals N, T and P which we will use to denote non-terminal alphabet, terminal alphabet and a set of productions of a grammar, respectively.

3.1.2 Basic definitions

A Chomsky grammar is a quadruple $G = \{N, T, P, S\}$, where N is a nonterminal alphabet, T is an alphabet of terminals, P is a set of *productions*, or *rewriting rules*, written as $\lambda \rightarrow \mu$, and $S \in N$ is a starting symbol, or *axiom*. In further discussion V_G will denote $T \cup N$, and

V_G^* will stand for $(T \cup N)^*$ (zero or more elements of V_G). The complexity and restrictions on the grammar are primarily imposed by the way the rewriting rules are formulated.

By Chomsky's classification, we call a grammar *context-sensitive* (CSG) if P is a set of rules of the form $\lambda A \mu \rightarrow \lambda \omega \mu$ and λ, μ, ω being strings over V_G, $A \in N$, $\omega \neq \varepsilon$.

A grammar is *context-free* (CFG) when all its rules are of the form $A \rightarrow \lambda$, where $A \in N$, $\lambda \in V_G^*$.

A context-free grammar that has at most one non-terminal symbol on the right hand side, that is if $|\lambda|_N \leq 1$ it is called *linear*. Furthermore, if all rules $A \rightarrow \lambda$ of a context-free grammar have $\lambda \in T^*$ or $\lambda \in T^*N$, that is, if the rules have non-terminal symbol in the rightmost position, the grammar is said to be *right-linear*.

And, finally, a grammar is called *regular* (RG) when it contains only rules of form $A \rightarrow a$ or $A \rightarrow aB$, where $A, B \in N$, $a \in T$.

The language generated by the grammar G is denoted by $L(G)$. Two grammars are called *equivalent* if they generate the same language.

Rules of CFGs and RGs are often required to be presented in a special form - *Chomsky Normal Form* (CNF). CNF requires for all the rules to be written as either $A \rightarrow BC$ or $A \rightarrow a$. Every CFG and RG has a corresponding equivalent grammar in Chomsky Normal Form.

There exist many modifications to the Context-Free Grammars in their pure form, which allow the model designer to enforce some additional constraints on the model, typically available in the lower Chomsky level grammar mechanism[1]. Often, these extensions are employed when only limited capabilities of the less constrained grammar are needed, and there is no need to implement the corresponding machine, which is in general more computationally complex and slow.

3.1.3 Expressive Power of Language Model

The complexity and efficiency of the language representation is an important issue when modeling a process. If we have reasons to believe that the process under consideration bears features of some established level of complexity, it is most efficient to choose a model which

[1] Chomsky classification assigns level 0 to the least constrained free form grammar, realized by a general Turing Machine. More constrained grammars and languages, such as CSG, CFG, and RG are assigned to higher levels of the hierarchy (1, 2 and 3, respectively).

is the closest to the nature of the process and is the least complex. Chomsky language classification offers us an opportunity to make an informed decision as to what level of model complexity we need to employ to successfully solve the problem at hand.

Regular grammars

The regular grammar is the simplest form of language description. Its typical realization is an appropriate finite state machine which does not require memory to generate strings of the language. The absence of memory in the parsing/generating mechanism is the limiting factor on the language which the regular grammar represents. For instance, recursive structures cannot be captured by a regular grammar, since memory is required to represent the state of the parser before it goes into recursion in order to restore the state of the parsing mechanism after the recursive invocation. As was mentioned above, regular productions have the form: $A \rightarrow aB$ or $A \rightarrow a$. This form generates a language $L_r : a^n b^m \ldots$, where n and m are independent. In other words, knowing the value of one does not improve the knowledge of the other.

A finite automaton is not able to count, or match, any symbols in the alphabet, except to a finite number.

Context-free grammars

CFGs allow us to model dependencies between n and m in the language L_f consisting of strings like $a^n b^m \ldots$, which is generally referred to as *counting*. Counting in L_f should not be confused with counting in mathematical sense. In linguistics it refers to *matching* as opposed to some quantitative expression. That is, we can enforce a relation between n and m in L_f, but we cannot restrict it to have some fixed value, although it can be done by enumeration of the possibilities. Another subtle point of the CFG model is that the counting is recursive, which means that each symbol b matches symbols a in reverse order. That is, by using subscripts to denote order, L_f is, in fact, $a_1 a_2 \ldots a_n b_n \ldots b_2 b_1 \ldots$. Typically, CFG is realized as a push-down automaton (PDA), allowing for recursive invocations of the grammar sub-models (non-terminals).

Push-down automata cannot model dependencies between more than two elements of the alphabet (both terminal and non-terminal), that is, a language $a^n b^n c^n$ is not realizable by a PDA.

Context-sensitive grammars

The main difference between CFGs and CSGs is that no production in a CFG can affect the symbols in any other non-terminal in the string[2]. Context-sensitive grammars (CSG) have an added advantage of arbitrary order counting. In other words, a context-sensitive language L_s is such that $a^n b^m \ldots$ can be matched in any order. It is realizable by linearly-bound automaton[3] and is significantly more difficult to realize than CFG. Due to computational complexity associated with LBAs, limited context sensitivity is often afforded by extended, or special forms of CFG.

Examples

Let us, without further explanations, list some examples of languages of different levels of Chomsky hierarchy:

Regular	$L_1 = \{a^n b^m\}$
Context-Free	$L_2 = \{\mu a \mu^{-1}\}$, where μ^{-1} designates reverse order of μ $L_3 = \{a^n b^m c^m d^n\}$ $L_4 = \{a^n b^n\}$
Context-Sensitive	$L_5 = \{\mu a \mu\}$ $L_6 = \{a^n b^m c^n d^m\}$ $L_7 = \{a^n b^n c^n\}$

3.2 Stochastic Context-Free Grammars

The probabilistic aspect is introduced into syntactic recognition tasks via Stochastic Context-Free Grammars. A *Stochastic Context-Free Grammar* (SCFG) is a probabilistic extension of a Context-Free Grammar. The extension is implemented by adding a probability measure to every production rule:

$$A \to \lambda \quad [p]$$

The rule probability p is usually written as $P(A \to \lambda)$. This probability is a conditional

[2] In probabilistic framework, context freeness of a grammar translates into statistical independence of its non-terminals.

[3] Essentially, a Turing machine with finite tape.

probability of the production being chosen, given that non-terminal A is up for expansion (in generative terms). Saying that stochastic grammar is context-free essentially means that the rules are conditionally independent and, therefore, the probability of the complete derivation of a string is just a product of the probabilities of rules participating in the derivation.

Given a SCFG G, let us list some basic definitions:

1. The *probability of partial derivation* $\nu \Rightarrow \mu \Rightarrow \ldots \lambda$ is defined in inductive manner as
   (a) $P(\nu) = 1$
   (b) $P(\nu \Rightarrow \mu \Rightarrow \ldots \lambda) = P(A \rightarrow \omega)P(\mu \Rightarrow \ldots \lambda)$, where production $A \rightarrow \omega$ is a production of G, μ is derived from ν by replacing one occurrence of A with ω, and $\nu, \mu, \ldots, \lambda \in V_G^*$.
2. The *string probability* $P(A \Rightarrow^* \lambda)$ (Probability of λ given A) is the sum of all left-most derivations $A \Rightarrow \ldots \Rightarrow \lambda$.
3. The *sentence probability* $P(S \Rightarrow^* \lambda)$ (Probability of λ given G) is the string probability given the axiom S of G. In other words, it is $P(\lambda|G)$.
4. The *prefix probability* $P(S \Rightarrow_L^* \lambda)$ is the sum of the strings having λ as a prefix,

$$P(S \Rightarrow_L^* \lambda) = \sum_{\omega \in V_G^*} P(A \Rightarrow^* \lambda\omega)$$

   In particular, $P(S \Rightarrow_L^* \epsilon) = 1$.

3.3 Earley-Stolcke Parsing Algorithm

The method most generally and conveniently used in stochastic parsing is based on an Earley parser ([16]), extended in such a way as to accept probabilities.

In parsing stochastic sentences we adopt a slightly modified notation of [31]. The notion of a *state* is an important part of the Earley parsing algorithm. A state is denoted as:

$$i : X_k \rightarrow \lambda . Y \mu$$

where '.' is the marker of the current position in the input stream, i is the index of the marker, and k is the starting index of the substring denoted by nonterminal X. Nonterminal X is said to dominate substring $w_k...w_i...w_l$, where, in the case of the above state, w_l is the last terminal of substring μ.

In cases where the position of the dot and structure of the state is not important, for compactness we will denote a state as:

$$S_k^i \equiv i : X_k \rightarrow \lambda.Y\mu$$

Parsing proceeds as an iterative process sequentially switching between three steps - *prediction*, *scanning* and *completion*. For each position of the input stream, an Earley parser keeps a set of *states*, which denote all pending derivations. States produced by each of the parsing steps are called, respectively, *predicted*, *scanned* and *completed*. A state is called *complete* (not to be confused with *completed*), if the dot is located in the rightmost position of the state. A complete state is the one that "passed" the grammaticality check and can now be used as a "ground truth" for further abstraction. A state "explains" a string that it dominates as a possible interpretation of symbols $w_k...w_i$, "suggesting" a possible continuation of the string if the state is not complete.

3.3.1 Prediction

In the Earley parsing algorithm the prediction step is used to hypothesize the possible continuation of the input based on current position in the parse tree. Prediction essentially expands one branch of the parsing tree down to the set of its leftmost leaf nodes to predict the next possible input terminal. Using the state notation above, for each state S_k^i and production p of a grammar $G = \{T, N, S, P\}$ of the form

$$\begin{cases} S_k^i & : \quad i : X_k \rightarrow \lambda.Y\mu \\ p & : \quad Y \rightarrow \nu \end{cases} \tag{3.1}$$

where $Y \in N$, we predict a state

$$S_i^i : \quad i : Y_i \rightarrow .\nu$$

Prediction step can take on a probabilistic form by keeping track of the probability of choosing a particular predicted state. Given the statistical independence of the nonterminals of the SCFG, we can write the probability of predicting a state S_i^i as conditioned on probability of S_k^i. This introduces a notion of forward probability, which has the interpretation similar to that of a forward probability in HMMs. In SFCG the forward probability α_i is the probability of the parser accepting the string $w_1...w_{(i-1)}$ and choosing state S at a step i. To continue the analogy with HMMs, inner probability, γ_i, is a probability of generating a substring of the input from a given nonterminal, using a particular production. Inner probability is thus conditional on the presence of a nonterminal X with expansion started at the position k, unlike the forward probability, which includes the generation history starting with the axiom. Formally, we can integrate computation of α and γ with non-probabilistic Earley predictor as follows:

$$\begin{cases} i : X_k \rightarrow \lambda.Y\mu \quad [\alpha, \gamma] \\ Y \rightarrow \nu \end{cases} \quad \Longrightarrow \quad i : Y \rightarrow .\nu \quad [\alpha', \gamma']$$

where α' is computed as a sum of probabilities of all the paths, leading to the state $i : X_k \rightarrow \lambda.Y\mu$ multiplied by the probability of choosing the production $Y \rightarrow \nu$, and γ' is the rule probability, seeding the future substring probability computations:

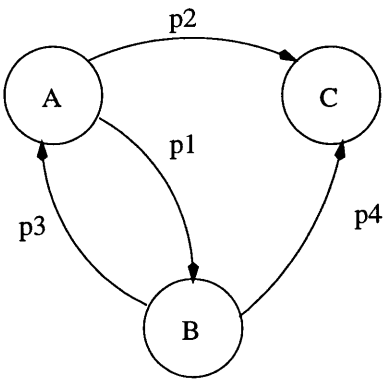
$$\begin{aligned} \alpha' &= \textstyle\sum_{\forall \lambda, \mu} \alpha(i : X_k \rightarrow \lambda.Y\mu) P(Y \rightarrow \nu) \\ \gamma' &= P(Y \rightarrow \nu) \end{aligned}$$

Recursive correction

Because of possible left recursion in the grammar the total probability of a predicted state can increase with it being added to a set infinitely many times. Indeed, if a non-terminal A is considered for expansion and given productions for A

$$\begin{aligned} A &\rightarrow Aa \\ &\rightarrow a \end{aligned}$$

we will predict $A \rightarrow .a$ and $A \rightarrow .Aa$ at the first prediction step. This will cause the predictor to consider these newly added states for possible descent, which will produce the same two states again. In non-probabilistic Earley parser it normally means that no further expansion of the production should be made and no duplicate states should be added. In



G_1:

$$
\begin{array}{lcll}
A & \rightarrow & BB & [p_1] \\
 & \rightarrow & CB & [p_2] \\
B & \rightarrow & AB & [p_3] \\
 & \rightarrow & C & [p_4] \\
C & \rightarrow & a & [p_5]
\end{array}
\qquad
\left|\begin{array}{ccc}
0 & p_1 & p_2 \\
p_3 & 0 & p_4 \\
0 & 0 & 0
\end{array}\right|
$$

Figure 3-1: Left Corner Relation graph of the grammar G_1. Matrix P_L is shown on the right of the productions of the grammar.

probabilistic version, however, although adding a state will not add any more information to the parse, its probability has to be factored in by adding it to the probability of the previously predicted state. In the above case that would mean an infinite sum due to left-recursive expansion.

In order to demonstrate the solution we first need to introduce the concept of Left Corner Relation.

Two nonterminals are said to be in a Left Corner Relation $X \rightarrow_L Y$ *iff there exists a production for* X *of the form* $X \rightarrow Y\lambda$.

We can compute a total recursive contribution of each left-recursive production rule where nonterminals X and Y are in Left Corner Relation. The necessary correction for non-terminals can be computed in a matrix form. The form of recursive correction matrix R_L can be derived using simple example presented in Figure 3-1. The graph of the relation presents direct left corner relations between nonterminals of G_1. The LC Relationship matrix P_L of the grammar G_1 is essentially the adjacency matrix of this graph. If there exists an edge between two nonterminals X and Y it follows that the probability of emitting terminal a, such that

$$
\begin{cases}
X \rightarrow Y\nu \\
Y \rightarrow a\eta
\end{cases}
$$

G_2:

S	$\rightarrow$	AB	$[p_1]$
	$\rightarrow$	C	$[p_2]$
	$\rightarrow$	d	$[p_3]$
A	$\rightarrow$	AB	$[p_4]$
	$\rightarrow$	a	$[p_5]$
B	$\rightarrow$	bB	$[p_6]$
	$\rightarrow$	b	$[p_7]$
C	$\rightarrow$	BC	$[p_8]$
	$\rightarrow$	B	$[p_9]$
	$\rightarrow$	c	$[p_{10}]$

P_L	S	A	B	C
S		p_1		p_2
A		p_4		
B				
C			$p_8 + p_9$	

R_L	S	A	B	C
S	1	$\frac{-p_1}{-1+p_4}$	$p_2p_8 + p_2p_9$	p_2
A		$\frac{-1}{-1+p_4}$		
B			1	
C			$p_8 + p_9$	1

Table 3.1: Left Corner (P_L) and its Reflexive Transitive Closure (R_L) matrices for a simple SCFG.

(from which follows that $X \rightarrow a\eta\nu$) is a sum of probabilities along all the paths connecting X with Y, multiplied by probability of direct emittance of a from Y, $P(Y \rightarrow a\eta)$.

Formally,

$$\begin{aligned} P(a) &= P(Y \rightarrow a\eta) \sum_{\forall X} P(X \Rightarrow^* Y) \\ &= P(Y \rightarrow a\eta) (P_0(X \Rightarrow^* Y) + \\ &\qquad P_1(X \Rightarrow^* Y) + \\ &\qquad P_2(X \Rightarrow^* Y) + \ldots \end{aligned}$$

where $P_k(X \Rightarrow Y)$ is probability of a path from X to Y of length $k = 1, \ldots, \infty$

In matrix form all the $k-$step path probabilities on a graph can be computed from its adjacency matrix as P_L^k. And the complete reflexive transitive closure R_L is a matrix of infinite sums, that has a closed form solution:

$$R_L = P_L^0 + P_L^1 + P_L^2 + \ldots = \sum_{k=0}^{\infty} P_L^k = (I - P_L)^{-1}$$

Thus, the correction to the completion step should be applied as first, descending the chain of left corner productions, indicated by a non-zero value in R_U:

$$\begin{cases} i : X_k \rightarrow \lambda . Z\mu \quad [\alpha, \gamma] \\ \forall Z \quad s.t. \quad R_L(Z, Y) \neq 0 \quad \Longrightarrow \quad i : Y \rightarrow .\nu \quad [\alpha', \gamma'] \\ Y \rightarrow \nu \end{cases}$$

and then correcting the forward and inner probabilities for the left recursive productions by an appropriate entry of the matrix R_L:

$$\begin{aligned} \alpha' &= \sum_{\forall \lambda,\mu} \alpha(i : X_k \rightarrow \lambda.Z\mu) R_L(Z,Y) P(Y \rightarrow \nu) \\ \gamma' &= P(Y \rightarrow \nu) \end{aligned}$$

Matrix R_L can be computed once for the grammar and used at each iteration of the prediction step.

3.3.2 Scanning

Scanning step simply reads the input symbol and matches it against all pending states for the next iteration. For each state $X_k \rightarrow \lambda.a\mu$ and the input symbol a we generate a state $i+1 : X_k \rightarrow \lambda a.\mu$:

$$i : X_k \rightarrow \lambda.a\mu \implies i+1 : X_k \rightarrow \lambda a.\mu$$

It is readily converted to the probabilistic form. The forward and inner probabilities remain unchanged from the state being confirmed by the input, since no selections are made at this step. The probability, however, may change if there is a likelihood associated with the input terminal. This will be exploited in the next chapter dealing with uncertain input. In probabilistic form the scanning step is formalized as:

$$i : X_k \rightarrow \lambda.a\mu \quad [\alpha,\gamma] \implies i+1 : X_k \rightarrow \lambda a.\mu \quad [\alpha,\gamma]$$

where α and γ are forward and inner probabilities.

Note the increase in i index. This signifies the fact that scanning step inserts the states into the new state set for the next iteration of the algorithm.

3.3.3 Completion

Completion step, given a set of states which just have been confirmed by scanning, updates the positions in all the pending derivations all the way up the derivation tree. The position in expansion of a pending state $j : X_k \rightarrow \lambda.Y\mu$ is advanced if there is a state, starting at position j, $i : Y_j \rightarrow \nu.$, which consumed all the input symbols related to it. Such a state can now be used to confirm other states, expecting Y as their next non-terminal. Since the

index range is attached to Y, we can effectively limit the search for the pending derivations to the state set, indexed by the starting index of the completing state, j:

$$\begin{cases} j : X_k \to \lambda.Y\mu \\ i : Y_j \to \nu. \end{cases} \Longrightarrow i : X_k \to \lambda Y.\mu$$

Again, propagation of forward and inner probabilities is simple. New values of α and γ are computed by multiplying the corresponding probabilities of the state being completed by the total probability of all paths, ending at $i : Y_j \to \nu.$. In its probabilistic form, completion step generates the following states:

$$\begin{cases} j : X_k \to \lambda.Y\mu \quad [\alpha, \gamma] \\ i : Y_j \to \nu. \quad [\alpha'', \gamma''] \end{cases} \Longrightarrow i : X_k \to \lambda Y.\mu \quad [\alpha', \gamma']$$

$$\begin{aligned} \alpha' &= \textstyle\sum_{\forall\lambda,\mu} \alpha(i : X_k \to \lambda.Y\mu)\gamma''(i : Y_j \to \nu.) \\ \gamma' &= \textstyle\sum_{\forall\lambda,\mu} \gamma(i : X_k \to \lambda.Y\mu)\gamma''(i : Y_j \to \nu.) \end{aligned}$$

Recursive correction

Here, as in prediction, we might have a potential danger of entering an infinite loop. Indeed, given the productions

$$\begin{aligned} A &\to \text{B} \\ &\to \text{a} \\ B &\to \text{A} \end{aligned}$$

and the state $i : A_j \to a.$, we complete the state set j, containing:

$$\begin{aligned} j : A_j &\to \ . \ \text{B} \\ j : A_j &\to \ . \ \text{a} \\ j : B_j &\to \ . \ \text{A} \end{aligned}$$

Among others this operation will produce another state $i : A_j \to a.$, which will cause the parse to go into infinite loop. In non-probabilistic Earley parser that would mean that we just simply do not add the newly generated states and proceed with the rest of them. However, this will introduce the truncation of the probabilities as in the case with prediction. It has been shown that this infinite loop appears due to so-called *unit productions* ([32]).

G_2:

S	$\rightarrow$	AB	$[p_1]$
	$\rightarrow$	C	$[p_2]$
	$\rightarrow$	d	$[p_3]$
A	$\rightarrow$	AB	$[p_4]$
	$\rightarrow$	a	$[p_5]$
B	$\rightarrow$	bB	$[p_6]$
	$\rightarrow$	b	$[p_7]$
C	$\rightarrow$	BC	$[p_8]$
	$\rightarrow$	B	$[p_9]$
	$\rightarrow$	c	$[p_{10}]$

P_U	S	A	B	C
S				p_2
A				
B				
C			p_9	

R_U	S	A	B	C
S	1		p_2p_9	p_2
A		1		
B			1	
C			p_9	1

Table 3.2: Unit Production (P_U) and its Reflexive Transitive Closure (R_U) matrices for a simple SCFG.

Two nonterminals are said to be in a Unit Production Relation $X \rightarrow_U Y$ iff there exists a production for X of the form $X \rightarrow Y$.

As in the case with prediction we compute the closed-form solution for a Unit Production recursive correction matrix R_U (figure 3.2), considering the Unit Production relation, expressed by a matrix P_U. R_U is found as $R_U = (I - P_U)^{-1}$. The resulting expanded completion algorithm accommodates for the recursive loops:

$$\begin{cases} j : X_k \rightarrow \lambda . Z\mu \quad [\alpha, \gamma] \\ \forall Z \quad s.t. \quad R_U(Z,Y) \neq 0 \implies i : X_k \rightarrow \lambda Z . \mu \quad [\alpha', \gamma'] \\ i : Y_j \rightarrow \nu . \quad [\alpha'', \gamma''] \end{cases}$$

where computation of α' and γ' is corrected by a corresponding entry of R_U:

$$\begin{aligned} \alpha' &= \textstyle\sum_{\forall \lambda, \mu} \alpha(i : X_k \rightarrow \lambda . Z\mu) R_U(Z,Y) \gamma''(i : Y_j \rightarrow \nu .) \\ \gamma' &= \textstyle\sum_{\forall \lambda, \mu} \gamma(i : X_k \rightarrow \lambda . Y\mu) R_U(Z,Y) \gamma''(i : Y_j \rightarrow \nu .) \end{aligned}$$

As R_L, unit production recursive correction matrix R_U can be computed once for each grammar.

3.4 Relation to HMM

While SCFGs look very different from HMMs, they have some remarkable similarities. In order to consider the similarities in more detail, let us look at a subset of SCFGs -

Stochastic Regular Grammars. SRG, as defined above (section 3.1, page 28), is a 4-tuple $G = N, T, P, S$, where the set of productions P is of the form $A \rightarrow aB$, or $A \rightarrow a$. If we consider the fact that HMM is a version of probabilistic Finite State Machine, we can apply simple non-probabilistic rules to convert an HMM into an SRG. More precisely, taking transition $A \rightarrow^a B$, with probability p_m of an HMM, we can form an equivalent production $A \rightarrow aB \quad [p_m]$. Conversion of an SCFG in its general form is not always possible. An SCFG rule $A \rightarrow aAa$, for instance cannot be converted to a sequence of transitions of an HMM. Presence of this mechanism makes richer structural derivations, such as counting and recursion, possible and simple to formalize.

Another, more subtle difference has its roots in the way HMMs and SCFGs assign probabilities. SCFGs assign probabilities to the production rules, so that for each non-terminal, the probabilities sum up to 1. It translates into probability being distributed over sentences of the language - probabilities of all the sentences (or structural entities) of the language sum up to 1. HMMs in this respect are quite different - the probability gets distributed over all the sequences of a length n ([15]).

In the HMM framework, the tractability of the parsing is afforded by the fact that the number of states remains constant throughout the derivation. In the Earley SCFG framework, the number of possible Earley states increases linearly with the length of the input string, due to the starting index of each state. This makes it necessary in some instances to employ pruning techniques which allow us to deal with increasing computational complexity when parsing long input strings.

3.5 Viterbi Parsing

Viterbi parse is applied to the state sets in a chart parse in a manner similar to HMMs. Viterbi probabilities are propagated in the same way as inner probabilities, with the exception that instead of summing the probabilities during completion step, maximization is performed. That is, given a complete state S_j^i, we can formalize the process of computing Viterbi probabilities v_i as follows:

$$v_i(S_k^i) = max_{S_j^i}(v_i(S_j^i)v_j(S_k^j))$$

and the Viterbi path would include the state

$$S_k^i = argmax_{S_j^i}(v_i(S_j^i)v_j(S_k^j))$$

The state S_k^i keeps a back-pointer to the state S_j^i, which completes it with maximum probability, providing the path for backtracking after the parse is complete. The computation proceeds iteratively, within the normal parsing process. After the final state is reached, it will contain pointers to its immediate children, which can be traced to reproduce the maximum probability derivation tree.

3.5.1 Viterbi Parse in Presence of Unit Productions

The otherwise straight forward algorithm of updating Viterbi probabilities and the Viterbi path is complicated in the completion step by the presence of the Unit productions in the derivation chain. Computation of the forward and inner probabilities has to be performed in closed form, as described previously, which causes the unit productions to be eliminated from the Viterbi path computations. This results in producing an inconsistent parse tree with omitted unit productions, since in computing the closed form correction matrix, we collapse such unit production chains. In order to remedy this problem, we need to consider two competing requirements to the Viterbi path generation procedure.

1. On one hand, we need a batch version of the probability calculation performed, as previously described, to compute transitive recursive closures on all the probabilities, by using recursive correction matrices. In this case, Unit productions do not need to be considered by the algorithm, since their contributions to the probabilities are encapsulated in the recursive correction matrix R_U.

2. On the other hand, we need a finite number of states to be considered as completing states in a natural order so that we can preserve the path through the parse to have a continuous chain of parents for each participating production. In other words, the unit productions, which get eliminated from the Viterbi path during the parse while computing Viterbi probability, need to be included in the complete derivation tree.

We solve both problems by computing the forward, inner and Viterbi probabilities in closed form by applying the recursive correction R_U to each completion for non-unit com-

pleting states and then, considering only completing unit production states for inserting them into the production chain. Unit production states will not update their parent's probabilities, since those are already correctly computed via R_U. Now the maximization step in computing the parent's Viterbi probability needs to be modified to keep a consistent tree. To accomplish this, when using the unit production to complete a state, the algorithm inserts the Unit production state into a child slot of the completed state only if it has the same children as the state it is completing. The overlapping set of children is removed from the parent state. It extends the derivation path by the Unit production state, maintaining consistent derivation tree.

Pseudo-code of the modifications to the completion algorithm which maintains the consistent Viterbi derivation tree in presence of Unit production states is shown in Figure 3-2.

However, the problem stated in subsection 3.3.3 still remains. Since we cannot compute the Viterbi paths in closed form, we have to compute them iteratively, which can make the parser go into an infinite loop. Such loops should be avoided while formulating the grammar. A better solution is being sought.

```
function StateSet.Complete()
        ...
    if(State.IsUnit())
        State.Forward = 0;
        State.Inner = 0;
    else
        State.Forward = ComputeForward();
        State.Inner = ComputeInner();
    end

    NewState = FindStateToComplete();
    NewState.AddChild(State);
    StateSet.AddState(NewState);
        ...
end


function StateSet.AddState(NewState)
        ...
    if(StateSet.AlreadyExists(NewState)
        OldState = StateSet.GetExistingState(NewState);
        CompletingState = NewState.GetChild();
        if(OldState.HasSameChildren(CompletingState))
                OldState.ReplaceChildren(CompletingState);
        end
        OldState.AddProbabilities(NewState.Forward, NewState.Inner);
    else
        StateSet.Add(NewState);
    end
        ...
end
```

Figure 3-2: Simplified pseudo-code of the modifications to the completion algorithm.

Chapter 4

Temporal Parsing of Uncertain Input

The approach described in the previous chapter can be effectively used to find the most likely syntactic groupings of the elements in a non-probabilistic input stream. The probabilistic character of the grammar comes into play when disambiguation of the string is required by the means of the probabilities attached to each rule. These probabilities reflect the "typicality" of the input string. By tracing the derivations, to each string we can assign a value, reflecting how typical the string is according to the current model.

In this chapter we to address two main problems in application of the parsing algorithm described so far to action recognition:

- Uncertainty in the input string.

  The input symbols which are formed by low level temporal feature detectors are uncertain. This implies that some modifications to the algorithm which account for this are necessary.

- Temporal consistency of the event stream.

  Each symbol of the input stream corresponds to some time interval. Since here we are dealing with a single stream input, only one input symbol can be present in the stream at a time and no overlapping is allowed, which the algorithm must enforce.

4.1 Terminal Recognition

Before we proceed to develop the temporal parsing algorithm, we need to say a few words about the formation of the input. In this thesis we are attempting to combine the detection of independent component activities in a framework of syntax-driven structure recognition. Most often these components are detected "after the fact", that is, recognition of the activity only succeeds when the whole corresponding sub-sequence is present in the causal signal. At the point when the activity is detected by the low level recognizer, the likelihood of the activity model represents the probability that this part of the signal has been generated by the corresponding model. In other words, with each activity likelihood, we will also have the sample range of the input signal, or a corresponding time interval, to which it applies. This fact will be the basis of the technique for enforcing temporal consistency of the input, developed later in this chapter. We will refer to the model activity likelihood as a *terminal*, and to the length of corresponding sub-sequence as a *terminal length.*

4.2 Uncertainty in the Input String

The previous chapter described the parsing algorithm, where no uncertainty about the input symbols is taken into consideration. New symbols are read off by the parser during the scanning step, which changes neither forward nor inner probabilities of the pending states. If the *likelihood of the input symbol* is also available, as in our case, it can easily be incorporated into the parse during scanning by multiplying the inner and forward probability of the state being confirmed by the input likelihood. We reformulate the scanning step as follows: having read a symbol a and its likelihood $P(a)$ off the input stream, we produce the state

$$i : X_k \rightarrow \lambda.a\mu \quad [\alpha, \gamma] \implies i+1 : X_k \rightarrow \lambda a.\mu \quad [\alpha', \gamma']$$

and compute new values of α' and γ' as:

$$\begin{aligned} \alpha' &= \alpha(i : X_k \rightarrow \lambda.a\mu)P(a) \\ \gamma' &= \gamma(i : X_k \rightarrow \lambda.a\mu)P(a) \end{aligned}$$

The new values of forward and inner probabilities will weigh competing derivations not only by their typicality, but also by our certainty about the input at each sample.

4.3 Substitution

Introducing likelihoods of the terminals at the scanning step makes it simple to employ the *multi-valued string* parsing ([10]) where each element of the input is in fact a multi-valued vector of vocabulary model likelihoods at each time step of an event stream. Using these likelihoods, we can condition the maximization of the parse not only on frequency of occurrence of some sequence in the training corpus, but also on measurement or estimation of the likelihood of each of the sequence component in the test data.

The multi-valued string approach allows for dealing with so-called *substitution* errors which manifest themselves in replacement of a terminal in the input string with an incorrect one. It is especially relevant to the problems addressed by this thesis where sometimes the correct symbol is rejected due to a lower likelihood than that of some other one. In the proposed approach, we allow the input at discrete instance of time to include all non-zero likelihoods and to let the parser select the most likely one, based not only on the corresponding value, but also on the probability of the whole sequence, as additional reinforcement to the local model estimate.

From this point and on, the input stream will be viewed as a multi-valued string (a lattice) which has a vector of likelihoods, associated with each time step (e.g. figure 4-1). The length of the vector is in general equal to the number of vocabulary models. The model likelihoods are incorporated with the parsing process at the scanning step, as was shown above, by considering all non-zero likelihoods for the same state set, that is:

$$\begin{cases} i : X_k \rightarrow \lambda.a\mu \quad [\alpha, \gamma] \\ \forall a, \quad s.t. \quad P(a) > 0 \end{cases} \Longrightarrow i+1 : X_k \rightarrow \lambda a.\mu \quad [\alpha', \gamma']$$

The parsing is performed in a parallel manner for all suggested terminals.

4.4 Insertion

It has been shown that "while significant progress has been made in the processing of correct text, a practical system must be able to handle many forms of ill-formedness gracefully" [3]. In our case, the need for this robustness is paramount.

Indeed, the parsing needs to be performed on a lattice, where the symbols which we need to consider for inclusion in the string come at random times. This results in appearance

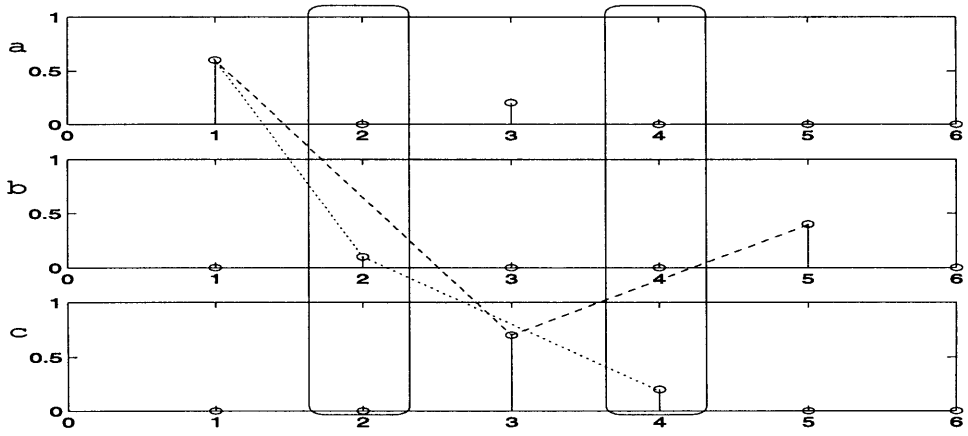


Figure 4-1: Example of the lattice parse input for three model vocabulary. Consider a grammar $A \rightarrow abc \mid acb$. The dashed line shows a parse *acb*. The two rectangles drawn around samples 2 and 4 show the "spurious symbols" for this parse which need to be ignored for the derivation to be contiguous. We can see that if the spurious symbols are simply removed from the stream, an alternative derivation for the sequence - *abc*, shown by a dotted line, will be interrupted. (Note the sample 3 which contains two concurrent symbols which are handled by the lattice parsing.)

of "spurious" (i.e. ungrammatical) symbols in the input stream, a problem commonly referred to as *insertion*. We need to be able to consider these inputs separately, at different time steps, and disregard them if their appearance in the input stream for some derivation is found to be ungrammatical. At the same time, we need to preserve the symbol in the stream for considering it in other possible derivations, perhaps even of a completely different string. The problem is illustrated by figure 4-1. In this figure we observe two independent derivations on an uncertain input stream. While deriving the string *acb*, connecting samples 1, 3 and 5, we need to disregard samples 2 and 4, which would interfere with the derivation. However we do not have an a priori information about the overall validity of this string, that is, we cannot simply discard samples 2 and 4 from the stream, because combined with sample 1, they form an alternative string, *abc*. For this alternative derivation, sample 3 would present a problem if not dealt with.

There are at least three possible solutions to the insertion problem in our context.

- Firstly, we can employ some *ad hoc* method which "synchronizes" the input, presenting the terminals coming with a slight spread around a synchronizing signal, as one vector. That would significantly increase the danger of not finding a parse if the spread is significant.

- Secondly, we may attempt gathering all the partially valid strings, performing all the derivations we can on the input and post-processing the resulting set to extract the most consistent maximum probability set of partial derivations. Partial derivations acquired by this technique will show a distinct split at the position in the input where the supposed ungrammaticality occurred. Being robust for finding ungrammatical symbols in a single stream, in our experiments, this method did not produce the desired results. The lattice extension, extremely noisy input and a relatively shallow grammar made it less useful, producing large number of string derivations, which contained a large number of splits and were difficult to analyze.

- And, finally, we can attempt to modify the original grammar to explicitly include the ERROR symbol (e.g. [2]).

In our algorithm the last approach proved to be the most suitable as it allowed us to incorporate some additional constraints on the input stream easily, as will be shown in the next section.

We formulate simple rules of the grammar modifications and perform the parsing of the input stream using this modified grammar.

Given the grammar G, robust grammar $\hat{G}$ is formed by following rules:

1. Each terminal in productions of G is replaced by a pre-terminal in $\hat{G}$:

   G : $\Rightarrow$ $\hat{G}$:

   $A \rightarrow bC$ $\quad$ $A \rightarrow \hat{B}C$

2. For each pre-terminal of $\hat{G}$ a skip rule is formed:

   $\hat{G}$:

   $\hat{B} \rightarrow b \mid SKIP\ b \mid b\ SKIP$

   This is essentially equivalent to adding a production $\hat{B} \rightarrow SKIP\ b\ SKIP$ if $SKIP$ is allowed to expand to ϵ.

3. Skip rule is added to $\hat{G}$, which includes all repetitions of all terminals:

   $\hat{G}$:

   $SKIP \rightarrow b \mid c \mid \ldots \mid b\ SKIP \mid c\ SKIP \mid \ldots$

Again, if $SKIP$ is allowed to expand to ϵ, the last two steps are equivalent to adding:

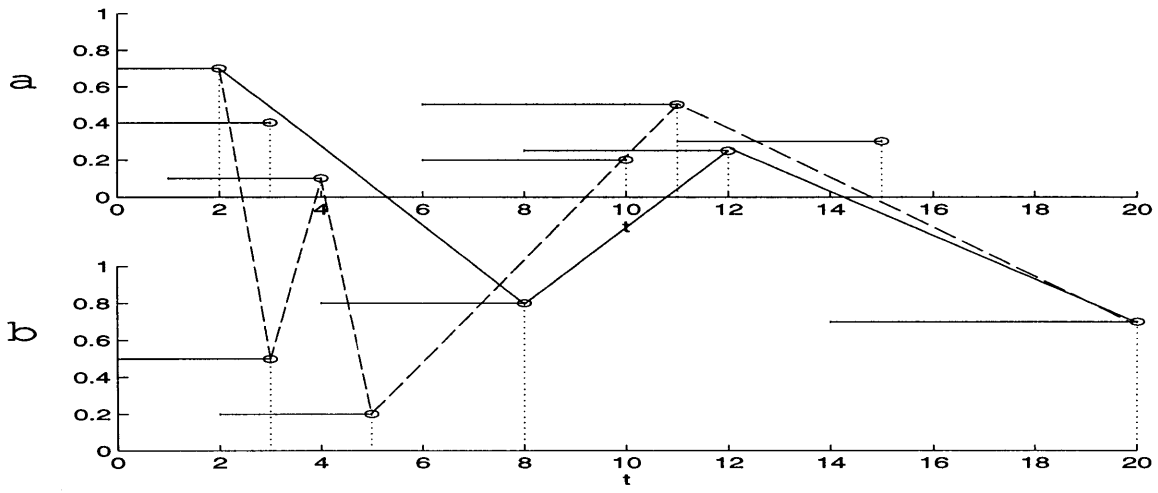


Figure 4-2: Example of temporally inconsistent terminals. Given a production $A \rightarrow ab \mid abA$ unconstrained parse will attempt to consume maximum amount of samples by non-$SKIP$ productions. The resulting parse, *ababab*, is shown by the dashed line. The solid line shows the correct parse, *abab*, which includes only non-overlapping terminals (horizontal lines show the sample range of each candidate terminal).

$\hat{G}$:

$\hat{B} \rightarrow SKIP \;\; b \;\; SKIP$

$SKIP \rightarrow \epsilon \mid b \;\; SKIP \;\; \ldots$

This process can be performed automatically as a pre-processing step, when the grammar is read in by the parser, so that no modifications to the grammar are explicitly written.

4.5 Enforcing Temporal Consistency

Parsing the lattice with the robust version of the original grammar results in the parser consuming a maximum amount of the input. Indeed, the $SKIP$ production, which is usually assigned a low probability, serves as a function that "penalizes" the parser for each symbol considered ungrammatical. This might result in the phenomenon where some of the "noise" gets forced to take part in a derivation as a grammatical occurrence. The effect of this can be relieved if we take into account the temporal consistency of the input stream, that is, only one event happens at a given time. Since each event has a corresponding length, we can further constrain the occurrence of the terminals in the input so that no "overlapping" events take place. The problem is illustrated in figure 4-2.

In Earley framework, we can modify the parsing algorithm to work in incremental fashion

as a filter to the completion step, while keeping track of the terminal lengths during scanning and prediction.

In order to accomplish the task we need to introduce two more state variables - h for "high mark" and l for "low mark". Each of the parsing steps has to be modified as follows:

1. **Scanning**

   Scanning step is modified to include reading the time interval associated with the incoming terminal. The updates of l and h are propagated during the scanning as follows:

   $$i : X_k \rightarrow \lambda.a\mu \quad [l, h] \quad \Longrightarrow \quad i+1 : X_k \rightarrow \lambda a.\mu \quad [l, h_a]$$

   where h_a is a high mark of the terminal. In addition to this, we set the time-stamp of the whole new $(i+1)$-th state set to h_a: $S_t = h_a$, to be used later by prediction step.

2. **Completion**

   Similarly to scanning, completion step advances the high mark of the completed state to that of the completing state, thereby extending the range of the completed non-terminal.

   $$\begin{cases} j : X_k \rightarrow \lambda.Y\mu \quad [l1, h1] \\ i : Y_j \rightarrow \nu. \quad [l2, h2] \end{cases} \quad \Longrightarrow \quad i : X_k \rightarrow \lambda Y.\mu \quad [l_1, h_2]$$

   This completion is performed for all states $i : Y_j \rightarrow \nu.$ subject to constraints $l_2 \geq h_1$ and $Y, X \neq SKIP$.

3. **Prediction**

   Prediction step is responsible for updating the low mark of the state to reflect the timing of the input stream.

   $$\begin{cases} i : X_k \rightarrow \lambda.Y\mu \\ Y \rightarrow \nu \end{cases} \quad \Longrightarrow \quad i : Y \rightarrow .\nu \quad [S_t, S_t]$$

   Here, S_t is the time-stamp of the state set, updated by the scanning step.

The essence of this filtering is that only the paths that are consistent in the timing of their terminal symbols are considered. This does not interrupt the parses of the sequence since filtering, combined with robust grammar, leaves the subsequences which form the parse connected via the skip states.

4.5.1 Misalignment Cost

In the process of temporal filtering, it is important to remember that the terminal lengths are themselves uncertain. A useful extension to the temporal filtering is to implement a softer penalty function, rather than a hard cut-off of all the overlapping terminals, as described above. It is easily achieved at the completion step where the filtering is replaced by a weighting function $f(\theta)$, of a parameter $\theta = h_1 - l_2$, which is multiplied into forward and inner probabilities. For instance, $f(\theta) = e^{-\psi\theta^2}$, where ψ is a penalizing parameter, can be used to weigh "overlap" and "spread" equally. The resulting penalty is incorporated with computation of α and γ:

$$\begin{aligned} \alpha' &= f(\theta)\textstyle\sum_{\forall\lambda,\mu}\alpha(i:X_k \rightarrow \lambda.Z\mu)R_U(Z,Y)\gamma''(i:Y_j \rightarrow \nu.) \\ \gamma' &= f(\theta)\textstyle\sum_{\forall\lambda,\mu}\gamma(i:X_k \rightarrow \lambda.Y\mu)R_U(Z,Y)\gamma''(i:Y_j \rightarrow \nu.) \end{aligned}$$

More sophisticated forms of $f(\theta)$ can be formulated to achieve arbitrary spread balancing as needed.

4.6 On-line Processing

4.6.1 Pruning

As we gain descriptive advantages with using SCFGs for recognition, the computational complexity, as compared to Finite State Models, increases. The complexity of the algorithm grows linearly with increasing length of the input string. This presents us with the necessity in certain cases of pruning low probability derivation paths. In Earley framework such a pruning is easy to implement. We simply rank the states in the state set according to their *per sample* forward probabilities and remove those that fall under a certain threshold. However, performing pruning one should keep in mind that the probabilities of the remaining states will be underestimated.

4.6.2 Limiting the Range of Syntactic Grouping

Context-free grammars have a drawback of not being able to express countable relationships in the string by a hard bound by any means other than explicit enumeration (the situation similar to the "secretary problem" with finite state networks). One implication

of this drawback is that we cannot formally limit the range of the application of a $SKIP$ productions and formally specify how many $SKIP$ productions can be included in a derivation for the string to still be considered grammatical. When parsing with robust grammar is performed, some of the non-terminals being considered for expansion cover large portions of the input string consuming only a few samples by non-$SKIP$ rules. The states, corresponding to such non-terminals, have low probability and typically have no effect on the derivation, but are still considered in further derivations, increasing the computational load. This problem is especially relevant when we have a long sequence and the task is to find a smaller subsequence inside it which is modeled by an SCFG. Parsing the whole sequence might be computationally expensive if the sequence is of considerable length. We can attempt to remedy this problem by limiting the range of productions.

In our approach, when faced with the necessity to perform such a parse, we use implicit pruning by operating a parser on a relatively small sliding window. The parse performed in this manner has two important features:

- The "correct" string, if exists, *ends* at the current sample.
- The *beginning* sample of such a string is unknown, but is within the window.

From these observations, we can formalize the necessary modifications to the parsing technique to implement such a sliding window parser:

1. Robust grammar should only include the $SKIP$ productions of the form $A \rightarrow a \mid SKIP\, a$ since the end of the string is at the current sample, which means that there will not be trailing noise, which is normally accounted for by a production $A \rightarrow a\ SKIP$.
2. Each state set in the window should be seeded with a starting symbol. This will account for the unknown beginning of the string. After performing a parse for the current time step, Viterbi maximization will pick out the maximum probability path, which can be followed back to the starting sample exactly.

This technique is equivalent to a run-time version of Viterbi parsing used in HMMs ([14]). The exception is that no "backwards" training is necessary since we have an opportunity to "seed" the state set with an axiom at an arbitrary position.

Chapter 5

Implementation and System Design

The recognition system is built as a two-level architecture, according to the two-level Bayesian model described in section 1.1.2, page 11. The system is written in C++, using a LAPAC-based math library. It runs on a variety of platforms - the tests were performed in HPUX, SGI IRIX, Digital UNIX, Linux and, in a limited way, MS Windows95/NT. The overall system architecture is presented in figure 5-1.

5.1 Component Level: HMM Bank

The first level of the architecture is an HMM bank, which consists of a set of independent HMMs, running in parallel on the input signal. To recognize the components of the model vocabulary, we train one HMM per primitive action component (terminal), using the traditional Baum-Welch re-estimation procedure. At run-time each of these HMMs performs a Viterbi parse ([23]) of the incoming signal and computes the likelihood of the action primitive. The run-time algorithm used by each HMM to recognize the corresponding word of the action vocabulary is a version of [14] which performs a "backward" match of the signal over a window of a reasonable size.

At each time step the algorithm performs the Viterbi parse of the part of the input signal covered by the window. The algorithm first inserts a new sample of the incoming signal into the first position of the input FIFO queue, advancing the queue and discarding the oldest sample. The Viterbi trellis is then recomputed to account for the new sample.

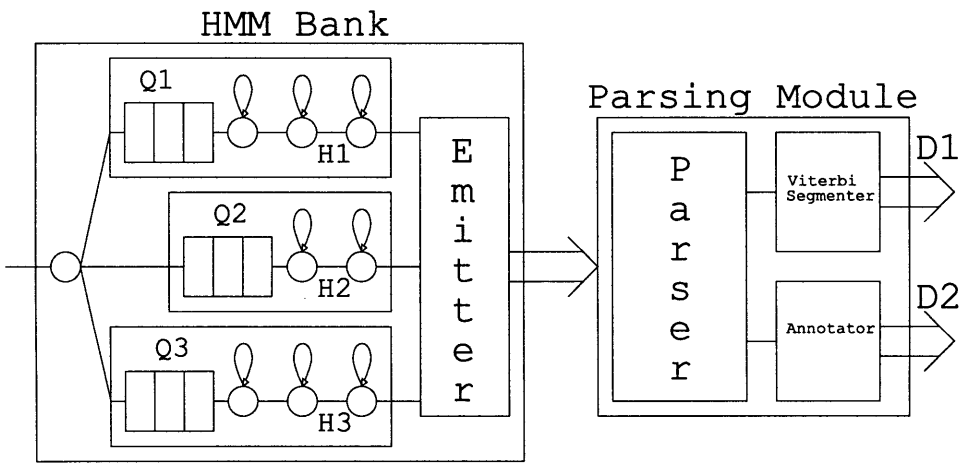


Figure 5-1: Architecture of the recognition system developed in the thesis. The system consists of an HMM bank and a parsing module. HMM bank contains the model HMMs ($H1$, $H2$, ...), operating independently on their own input queues ($Q1$, $Q2$, ...). The parsing module consists of a probabilistic parser, Viterbi parser, which performs segmentation and labeling (output data stream $D1$), and an annotation module (output data stream $D2$).

To find the maximum probability parse the trellis search is performed. Since the likelihood of the model decreases with the length of the sequence, the likelihood values in the trellis need to be normalized to express a *per sample* likelihood. This is expressed by a geometric mean, or, in case when the computations are performed in terms of log-likelihood, by a *log* geometric mean.

The maximum likelihood, found in the trellis, has a corresponding range of the input signal to which it applies. This value is easily extracted from the trellis length and the position of the maximum likelihood value in it. Figure 5-2 shows the output of a single HMM in a bank, illustrating the relation between output probabilities and sequence lengths.

All the HMMs are run in parallel, providing the parser with maximum normalized probability and the corresponding sequence length at each time step.

Event generation

The continuous vector output of the HMM bank is represented by a series of "events" which are considered for probabilistic parsing. It is in terms of these events, that the action grammar is expressed. In this step, we do not need to make strong decisions about discarding any events - we just generate a sufficient stream of evidence for the probabilistic parser such that it can perform structural rectification of these "suggestions". In particular the $SKIP$ states allow for ignoring erroneous events. As a result of such a rectification the parser

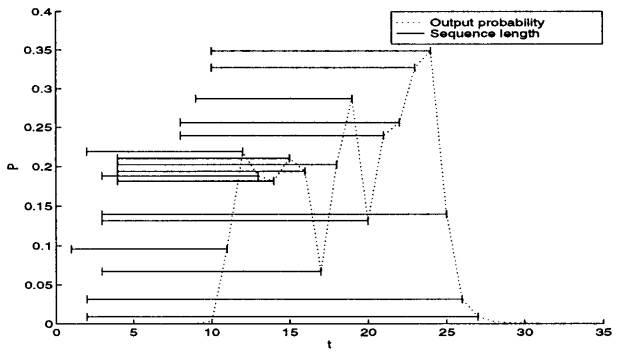


Figure 5-2: HMM output plot. Each point of the probability plot is the normalized maximum likelihood of the HMM at the current sample of the input signal. This value is found as a result of the backwards search on the FIFO queue, and corresponds to a sequence of a certain length. The plot shows those lengths as horizontal lines ending at the corresponding sample of the probability plot.

finds an interpretation of the stream which is structurally consistent (i.e. grammatical), is temporally consistent (uses non-overlapping sequence of primitives) and has maximum likelihood given the outputs of the low level feature detectors.

For the tasks discussed here, a simple discretization procedure provides good results. For each HMM in the bank, we select a very small threshold to cut off the noise in the output, and then search each of the areas of non-zero probabilities for a maximum[1]. Refer to figure 5-3a, for an example of the output of the HMM bank, superimposed with the results of discretization.

After discretization we replace the continuous time vector output of the HMM bank with the discrete symbol stream generated by discarding any interval of the discretized signal in which all values are zero. This output is emitted at run time, which requires a limited capability of the emitter to look back at the sequence of multi-valued observations. It accumulates the samples of each of the HMMs in a FIFO queue until it collects all the samples necessary to make the decision of the position of a maximum likelihood in all the model outputs. When the maxima are found, the corresponding batch of values is emitted into the output stream and the FIFO queue is flushed.

Figure 5-3b displays an example of the resulting event sequence generated.

[1] Alternatively, we can search the areas of non-zero likelihood for the position where the product of the probability and the sequence length is maximal. This will result in finding the weighted maximum, which corresponds to the maximum probability of the sequence of the maximum length.

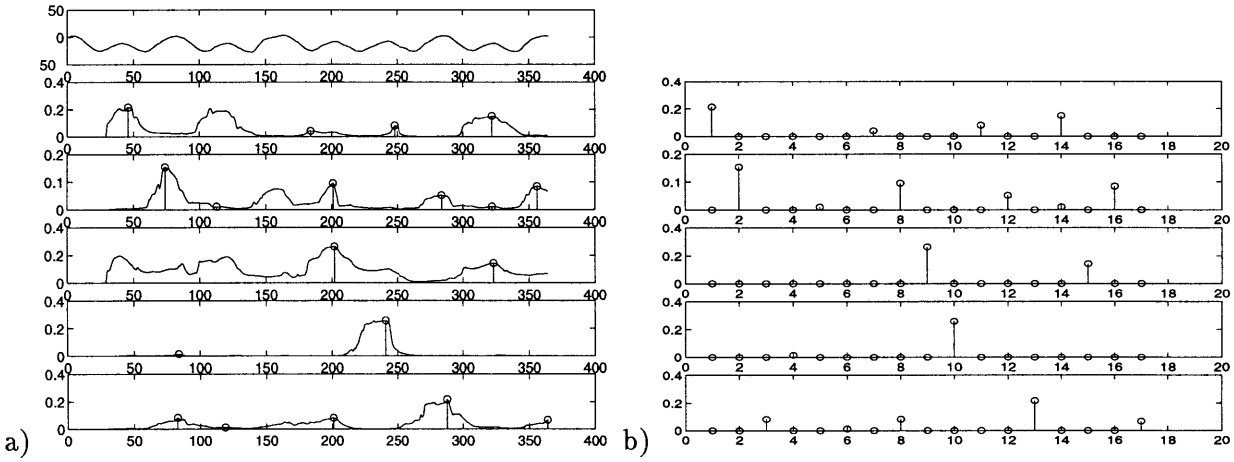


Figure 5-3: Example of the output of the HMM bank. a) Top plot shows the input sequence. Five plots below it show output of each HMM in the bank superimposed with the result of discretization. b) Corresponding discretized "compressed" sequence. Input vectors for the probabilistic parser are formed by taking the samples of all sequences for each time step.

5.2 Syntax Level: Parser

The probabilistic parser accepts input in form of an "event" sequence, generated by the HMM bank just described. Each sample of that sequence is a vector containing likelihoods of the candidate terminals and their corresponding lengths for the current time step.

5.2.1 Parser Structure

The parser encapsulates its functionality in three complete levels. The lowest *base level* is a complete non-probabilistic Earley parser which accepts an input string and parses it according to a given Context-Free Grammar. It implements all the basic functionality of the parsing algorithm in non-probabilistic setting.

A *stochastic extension* to the parser is built upon the basic Earley parser as a set of child classes, which derive all the basic functionality, but perform the parse according to a given Stochastic Context-Free Grammar. After the grammar is read and checked for consistency, the algorithm computes its recursive correction matrices R_L and R_U as was described in chapter 3. The parser implemented on this level is capable of performing a probabilistic parse of a string, which can be presented in either probabilistic or non-probabilistic form. Optionally, the parser can perform a *partial parse* where the most likely partial derivations are found if the string is ungrammatical.

A *multi-valued constrained parser* is derived from the stochastic extension and imple-

ments robust temporally consistent lattice parsing algorithm as described in chapter 4. It is at this level *sliding window* algorithm is also implemented.

5.2.2 Annotation Module

Annotation module is built into a Viterbi backtracking step. After the parse, the final state, if found, contains a back-pointer to the immediate children in a most likely derivation tree. Recursively traversing the tree in a depth-first manner, we essentially travel from one non-terminal to another in a natural order. This can be used for attaching an annotation to each of the non-terminals of interest.

The annotation is declared in the grammar file after each production for a non-terminal of interest. Annotation consists of text with embedded commands and formatting symbols and is emitted during the traversal of the Viterbi tree after the parse is completed. Every time a nonterminal which has an attached annotation is encountered, the annotation text is formatted according to the format specifications of the annotation block and is sent to the output.

A special mode of the annotation module implements the representative frame selection out of the video sequence, based on results of segmentation. When the range of a nonterminal is determined, the module computes frame indices for its starting and ending frames, after which it selects a representative frame based on the minimum distance to the mean image of the range.

Chapter 6

Experimental Results

The recognition system described in this thesis has been used to recognize several complex hand gestures. Each gesture has distinct components, forming its vocabulary, for which we trained the low-level temporal feature detectors (HMMs). The training data set consisted of 20 examples of gesture primitives. The resulting HMM bank was run on several test gestures, not present in the training set. For run time, each of the HMMs in the bank had its window and threshold parameters set manually. Output of the HMM bank is piped to the parser, which performed the parse and output the resulting Viterbi trace and annotated segmentation.

The rest of this section presents three types of experiments which were performed using the system.

6.1 Experiment I: Recognition and Disambiguation

The first experiment is simply a demonstration of the algorithm successfully recognizing a hand gesture and disambiguating its components in the global context. We define a `SQUARE` gesture as either left-handed (counterclockwise) or a right-handed (clockwise). The gesture vocabulary consists of four components - `left-right`, `up-down`, `right-left` and `down-up`. The interpretation of the gesture components is dependent on the global context. For instance, in the right-handed square, the `left-right` component is interpreted as the top of the square, whereas in the case of the left-handed square it is the bottom. We formulate a grammar for this model by introducing an additional non-terminal - `TOP`. This non-terminal well provide a label for `left-right` and `right-left` gestures based on the global context.

The grammar G_{square} provides a model which recognizes a SQUARE regardless of the fact that it can be either the right-handed or a left-handed square (with skip rules omitted for simplicity):

G_{square} :

SQUARE	→	RH	[0.5]
	|	LH	[0.5]
RH	→	TOP up-down right-left down-up	[0.25]
	|	up-down right-left down-up TOP	[0.25]
	|	right-left down-up TOP up-down	[0.25]
	|	down-up TOP up-down right-left	[0.25]
LH	→	left-right down-up TOP up-down	[0.25]
	|	down-up TOP up-down left-right	[0.25]
	|	TOP up-down left-right down-up	[0.25]
	|	up-down left-right down-up TOP	[0.25]
TOP	→	left-right	[0.5]
	→ |	right-left	[0.5]

We run the experiments first using the data produced by a mouse gesture, and then repeat the approach using the data from Polhemus sensor attached to the hand, and the data collected from the STIVE vision system ([36]). For each of the variations of the experiment we use separately trained set of HMMs.

Training data in all cases consists of 20 examples of each of the component gestures. 3-state HMMs, using x and y velocities as feature vectors, serve as models of those components and are trained on the set until the log-likelihood improvement falls below 0.001.

The testing is performed on a long unsegmented SQUARE gesture, performed in either right-handed or left-handed version. The new, previously unseen sequence, is segmented and labeled by the algorithm.

In the example shown in figure 6-1a the recovered structure was that of a right-hand square. Recognition results for a left-handed square sequence are seen in figure 6-1b. Note that the the label TOP was assigned to the left-right gesture in the first case and to the right-left in the second. The figures show sample indices, corresponding to each of the gesture primitives enclosed in square brackets. The indices, which were found during the

```
a) Segmentation <rsquare.dat>:
Label              Segment  LogP
---------------------------------------
Right hand square  [0   146] 8.619816e-01
        Top        [0   23 ] 8.557740e-01
        Up down    [23  66 ] 7.490584e-01
        Right left [66  94 ] 8.863638e-01
        Down up    [94  146] 9.567336e-01
Viterbi probability =  0.02400375


b) Segmentation <lsquare.dat>:
Label              Segment   LogP
---------------------------------------
Left hand square   [0   173] 8.304875e-01
        Left right [0   49 ] 9.351195e-01
        Down up    [49  71 ] 6.933256e-01
        Top        [71  132] 7.313213e-01
        Up down    [132 173] 9.593422e-01
Viterbi probability =  0.01651770
```

Figure 6-1: Example of the `SQUARE` sequence segmentation. In the presented example STIVE setup serves as the data source.
a) right-handed square segmentation, b) left-handed square segmentation.

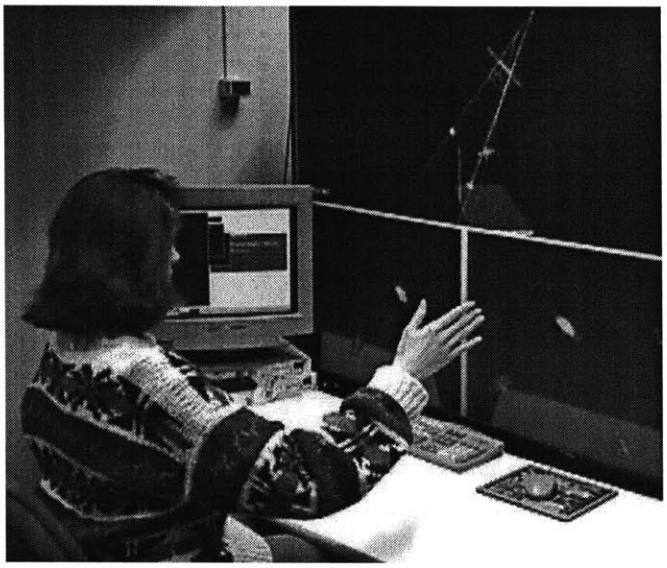


Figure 6-2: The Stereo Interactive Virtual Environment (STIVE) computer vision system used to collect data.

execution of the algorithm, form a continuous coverage of the input signal.

6.2 Recognition and semantic segmentation

As a more complex test of our approach we chose the domain of musical conducting. It is easy to think of conducting gestures as of complex gestures consisting of a combination of simpler parts for which we can write down a grammar (coincidentally, a book describing baton techniques, written by the famous conductor Max Rudolf [24] is called "The Grammar of Conducting"). We capture the data from a person, a trained conductor, using natural conducting gestures. The task we are attempting to solve is the following. A piece of music by Sibelius [1] includes a part with complex 6/4 music beat pattern. Instead of using this complex conducting pattern, conductors often use 2/4 or 3/4 gestures by merging 6/4 beats into groups of three or two at will. For the experiment we collect the data from a conductor conducting a few bars of the score arbitrarily choosing 2/4 or 3/4 gestures, and attempt to find bar segmentation, while simultaneously identifying the beat pattern used.

For the experiments we use Polhemus sensor and then apply the technique in the STIVE environment. We trained five HMMs on two primitive components of a 2/4 pattern and the three components of a 3/4 pattern. As a model we use a 4-state HMMs with x and y velocity

[1] Jean Sibelius (1865-1957), Second Symphony, Opus 43, in D Major

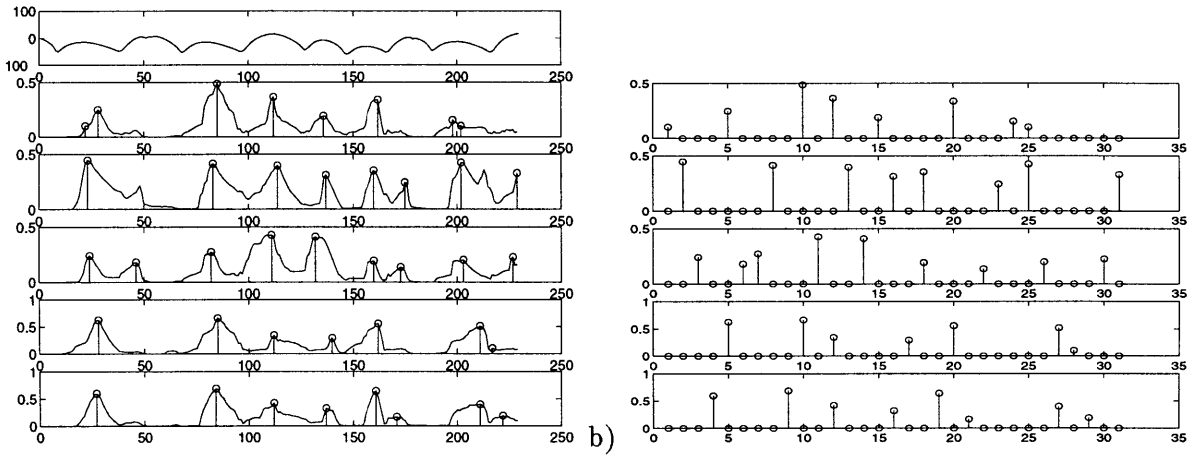


Figure 6-3: Output of the HMM bank for the sequence 2/4-2/4-3/4-2/4. a) Top plot shows the y - component of the input sequence. Five plots below it show output of each HMM in the bank superimposed with the result of discretization. b) Corresponding event sequence.

feature vectors for Polhemus data and a 6-state HMM with a feature vector containing x, y and z velocities of the hand for the STIVE experiment.

Some of the primitives in the set are very similar to each other and, therefore, corresponding HMMs show a high likelihood at about the same time, which results in a very "noisy" lattice (figure 6-3a). We parse the lattice with the grammar G_c (again, for simplicity omitting the SKIP productions):

G_c :

PIECE	$\rightarrow$	BAR PIECE	[0.5]
	|	BAR	[0.5]
BAR	$\rightarrow$	TWO	[0.5]
	|	THREE	[0.5]
THREE	$\rightarrow$	down3 right3 up3	[1.0]
TWO	$\rightarrow$	down2 up2	[1.0]

The results of run of a lower level part of the algorithm on a conducting sequence 2/4-2/4-3/4-2/4 are shown in figure 6-3 with the top plot of 6-3a displaying a y positional component.

Figure 6-4 demonstrates output of probabilistic parsing algorithm in the form of semantic labels and corresponding sample index ranges for the 2/4-2/4-3/4-2/4 gesture using Polhemus sensor. Results of the segmentation of a 2/4-3/4-2/4-3/4-2/4 gesture in the STIVE experiment are shown in figure 6-5.

```
Segmentation:
BAR:
    2/4     start/end sample: [0 66]
    Conducted as two quarter beat pattern.
BAR:
    2/4     start/end sample: [66 131]
    Conducted as two quarter beat pattern.
BAR:
    3/4     start/end sample: [131 194]
    Conducted as three quarter beat pattern.
BAR:
    2/4     start/end sample: [194 246]
    Conducted as two quarter beat pattern.
Viterbi probability =  0.00423416
```

Figure 6-4: Polhemus experiment: Results of the segmentation of a long conducting gesture for the bar sequence 2/4-2/4-3/4-2/4.

```
Segmentation:
BAR:
        2/4     start sample/end sample: [0 80]
        Conducted as two quarter bit pattern.
BAR:
        3/4     start sample/end sample: [80 210]
        Conducted as three quarter bit pattern.
BAR:
        2/4     start sample/end sample: [210 274]
        Conducted as two quarter bit pattern.
BAR:
        3/4     start sample/end sample: [274 404]
        Conducted as three quarter bit pattern.
BAR:
        2/4     start sample/end sample: [404 470]
        Conducted as two quarter bit pattern.
Viterbi probability =  0.00639809
```

Figure 6-5: STIVE experiment: results of the segmentation of a conducting gesture for the bar sequence 2/4-3/4-2/4-3/4-2/4.

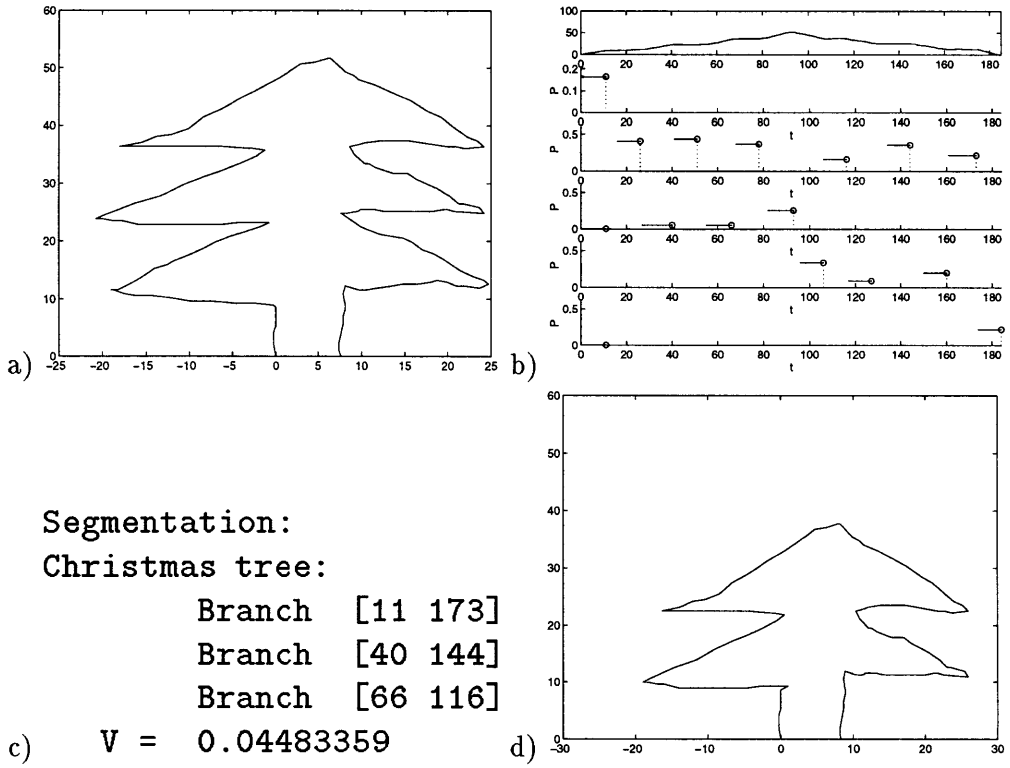


Figure 6-6: Labeling a recursive process. a) Christmas tree drawing as an example of a recursive process. b) Results of component recognition process with candidate events identified. Top plot shows the y component of the drawing plotted against time axis. c) Segmentation. d) Lower branch removed.

The segmentation and labeling achieved in these experiments are correct, which can be seen for one of the Polhemus experiments (refer to figures 6-3 and 6-4). In the interest of space we do not show intermediate results for the STIVE experiment, but it was also successful in all tests.

6.2.1 Recursive Model

The final experiment is presented here more for the sake of illustration of simplicity of describing a model for the recursive recognition task.

The task can be described as follows: we make a simple drawing of a tree as shown in figure 6-6a, starting at the left side of the trunk. We want to parse this drawing (presenting the data in the natural order), find the lowest branch and remove it from the drawing.

The drawing is described by a simple *recursive* grammar:

G_c :

TREE	$\rightarrow$	up BRANCH down	[1.0]
BRANCH	$\rightarrow$	LSIDE BRANCH RSIDE	[0.9]
	|	LSIDE RSIDE	[0.1]
LT-SIDE	$\rightarrow$	left up-diag	[1.0]
RT-SIDE	$\rightarrow$	dn-diag left	[1.0]

The intermediate results are shown in figure 6-6b. What is of interest in this example is the recursive character of the drawing and the fact that the definition of $BRANCH$ includes *long distance influences.* In other words, the part of the drawing which we want to label is composed of the primitives which are disparate in time, but one causes the other to occur. As seen from figure 6-6c, the algorithm has no problems with either of the difficulties.

The segmentation data now can be used to complete the task and remove the lower branch of the Christmas tree which is shown in figure 6-6d.

Chapter 7

Conclusions

7.1 Summary

In this thesis we implemented a recognition system based on a two-level recognition architecture for use in the domain of action recognition. The system consists of an HMM bank and a probabilistic Earley-based parser. In the course of the research we developed a set of extensions to the probabilistic parser which allow for uncertain multi-valued input and enforce temporal consistency of the input stream. An improved algorithm for building a Viterbi derivation tree was also implemented. The system includes an annotation module which is used to associate a semantic action to a syntactic grouping of the symbols of the input stream.

The recognition system performs syntactic segmentation, disambiguation and labeling of the stream according to a given Stochastic Context-Free Grammar and is shown to work in a variety of sensor modalities.

The use of formal grammars and syntax-based action recognition is reasonable if decomposition of the action under consideration into a set of primitives that lend themselves to automatic recognition is possible. We explored several examples of such actions and showed that the recognition goals were successfully achieved by the proposed approach.

7.2 Evaluation

The goals of this thesis and the means with which they are achieved in this work are summarized in the table below:

Goal	Means	Completed
Build a recognition system which is based on a simple description of a complex action as a collection of component primitives.	Achieved by a probabilistic Earley - based parser,a Stochastic Context-Free Grammar and extensions for temporal parsing.	√
Develop a segmentation technique based on a component content of the input signal.	Segmentation is performed as a structure probability maximization during the Viterbi parse.	√
Develop a context dependent annotation technique for an unsegmented input stream.	Annotation is emitted by a grammar driven annotation module during the structure probability maximization.	√

As is seen from the table, the main goals of the thesis have been reached.

Evaluating the contributions of this thesis the most important question to be answered is what this technique affords us. First of all, it allows us to assign probabilities to the structural elements of an observed action. Second of all, it allows for easy formulation of complex hierarchical action models. As an added advantage, it increases reliability of the estimation of the component primitive activities by filtering out the "impossible" observations and accommodates recursive actions.

Quantitative evaluation of the accuracy increase of the technique developed in this thesis is quite difficult to perform for two main reasons:

- Direct comparison of the current system with an unconstrained one involves building an alternative system which is quite an undertaking and it is still unclear which parameters of the system can be changed and which ones carried over. On the other hand, the evaluation problem can be simply stated as "not performing the parse". This alternative comparison is unfair since it is clearly biased to the system which we developed in this thesis.

- The improvement, afforded by additional syntactic constraints is problem specific. The essence of the syntactic filtering is that the full sequence space is significantly reduced by the grammar to a non-linear sub-space of admissible sequences, where the search is performed. The effectiveness of this approach is directly related to the decrease in the size of the search space. However, it is possible to write a grammar which covers the whole sequence space, in which case no improvement is afforded.

In general terms, the expected advantage of syntactic constraints was shown in introduction (figure 1-2). The figure shows that the syntactic constraints dramatically increase

recognition accuracy of the component primitives. This estimation is correct in our case, as was noted above.

7.3 Extensions

The recognition system described in this thesis can be easily extended to produce video annotations in the form of Web pages. It can be useful, for instance, for summarizing instructional videos.

It is possible to use Extended SCFGs with the current parser. Extended CFGs allow for limited context-sensitivity of the CFG parsing mechanism without opening the Pandora's box of Linearly bounded automata.

One interesting application of the window parsing algorithm, descried in section 4.6.2, is direct parsing of the outputs of an HMM bank with no explicit discretization. In this case, computational complexity can become prohibitive. However, if some knowledge about expected length of the signal is available, the original parsing task can be reformulated. The solution can be provided by splitting the original grammar into several components - one for each selected non-terminal and one which gathers the results into a final sentenial form. Each of the selected non-terminal models performs the parsing within some window of a limited size, thus, restricting the production range to a manageable size, as it is routinely done with HMMs. In this case, the sequence will effectively undergo a *semantic* discretization, which is then assembled by the top-level structure which has no window limit.

7.4 Final Thoughts

With all the advantages of the method there are still problems that were not addressed in our work, for instance, grammar inference is hard to do in the framework of stochastic parsing. We have not researched the opportunities of fully utilizing the production probabilities. In the experiments above we determined them heuristically using simple reasoning based on our understanding of the process. The rule probabilities, which reflect "typicality" of a string derivation, will play a significantly more important role in recognition and interpretation of actions of a higher complexity than those presented in the thesis. We plan on exploring the added advantages of the learned probabilities in our further research.

Bibliography

[1] A. V. Aho and Ullman J. D. *The Theory of Parsing, Translation and Compiling. Volume 1: Parsing.* Prentice Hall, Englewoods Cliffs, N.J., 1972.

[2] A. V. Aho and T. G. Peterson. A minimum distance error-correcting parser for context-free languages. *SIAM Journal of Computing*, 1, 1972.

[3] J. Allen. Special issue on ill-formed input. *American Journal of Computational Linguistics*, 9(3–4):123–196, 1983.

[4] M. L. Baird and M. D. Kelly. A paradigm for semantic picture recognition. *PR*, 6(1):61–74, 1974.

[5] A. F. Bobick. Movement, activity, and action: The role of knowledge in the perception of motion. In *Philosophical Transactions Royal Society London B*, 1997.

[6] A. F. Bobick and A. D. Wilson. A state-based technique for the summarization and recognition of gesture. *Proc. Int. Conf. Comp. Vis.*, 1995.

[7] T. L. Booth and R. A. Thompson. Applying probability measures to abstract languages. *IEEE Transactions on Computers*, c-22(5):442–450, May 1973.

[8] M. Brand. Understanding manipulation in video. In *AFGR96*, pages 94–99, 1996.

[9] R. A. Broks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, April 1989.

[10] H. Bunke and D. Pasche. Parsing multivalued strings and its application to image and waveform recognition. *Structural Pattern Analisys*, 1990.

[11] L. Campbell. Recognizing classical ballet setps using phase space constraints. Master's thesis, Massachusetts Institute of Technology, 1994.

[12] J. H. Connell. *A Colony Architecture for an Artificial Creature.* PhD thesis, MIT, 1989.

[13] J. D. Courtney. Automatic video indexing via object motion analysis. *PR*, 30(4):607–625, April 1997.

[14] T. J. Darrell and A. P. Pentland. Space-time gestures. *Proc. Comp. Vis. and Pattern Rec.*, pages 335–340, 1993.

[15] Charniak. E. *Statistical Language Learning.* MIT Press, Cambridge, Massachusetts and London, England, 1993.

[16] Jay Clark Earley. *An Efficient Context-Free Parsing Algorithm.* PhD thesis, Carnegie-Mellon University, Computer Science Department, Carnegie-Mellon University, Pittsburg, PA, Aug. 1968.

[17] K. S. Fu. A step towards unification of syntactic and statistical pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(3):398–404, May 1986.

[18] F. Jelinek, J. D. Lafferty, and R. L. Mercer. Basic methods of probabilistic context free grammars. In Pietro Laface and Renato Di Mori, editors, *Speech Recognition and Understanding. Recent Advances, Trends, and Applications*, volume F75 of *NATO Advanced Study Institute*, pages 345–360. Springer Verlag, Berlin, July 1992.

[19] P. Maes. The dynamic of action selection. *AAAI Spring Symposium on AI Limited Rationality*, 1989.

[20] R. Narasimhan. Labeling schemata and syntactic descriptions of pictures. *InfoControl*, 7(2):151–179, June 1964.

[21] B. J. Oomen and R. L. Kashyap. Optimal and information theoretic syntactic pattern recognition for traditional errors. *SSPR 6th International Workshop on Advances in Structural and Syntactic Pattern Recognition.*, 1996.

[22] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proc. of the IEEE*, 77(2):257–285, February 1989.

[23] L. R. Rabiner and B. H. Juang. *Fundamentals of speech recognition.* Prentice Hall, Englewood Cliffs, 1993.

[24] Max Rudolf. *The Grammar of Conducting. A Comprehensive Guide to Baton Techniques and Interpretation.* Schimmer Books, New York, 1994.

[25] A. Sanfeliu and R. Alquezar. Efficient recognition if a class of context-sensitive languages described by augmented regular expressions. *SSPR 6th International Workshop on Advances in Structural and Syntactic Pattern Recognition.*, 1996.

[26] A. Sanfeliu and M. Sainz. Automatic recognition of bidimensional models learned by grammatical inferrence in outdor scenes. *SSPR 6th International Workshop on Advances in Structural and Syntactic Pattern Recognition.*, 1996.

[27] R. Schalkoff. *Pattern Recognition: Statistical, Structural and Neural Approaches.* Wiley, New York, 1992.

[28] J. Schlenzig, E. Hunter, and R. Jain. Recursive identification of gesture inputs using hidden markov models. *Proc. Second Annual Conference on Applications of Computer Vision*, pages 187–194, December 1994.

[29] T. Starner and A. Pentland. Real-time american sign language from video using hidden markov models. In *MBR97*, page Chapter 10, 1997.

[30] T. E. Starner and A. Pentland. Visual recognition of American Sign Language using hidden markov models. In *Proc. of the Intl. Workshop on Automatic Face- and Gesture-Recognition*, Zurich, 1995.

[31] A. Stolcke. *Bayesian Learning of Probabilistic Language Models.* PhD thesis, University of California at Berkeley, 1994.

[32] A. Stolcke. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, 21(2), 1995.

[33] M. G. Thomason. Stochastic syntax-directed translation schemata for correction of errors in context-free languages. *TC*, 24:1211–1216, 1975.

[34] W. G. Tsai and K. S. Fu. Attributed grammars - a tool for combining syntatctic and statistical approaches to pattern recognition. In *IEEE Transactions on Systems, Man and Cybernetics*, volume SMC-10, number 12, pages 873–885, 1980.

[35] A. D. Wilson, A. F. Bobick, and J. Cassell. Temporal classification of natural gesture and application to video coding. *Proc. Comp. Vis. and Pattern Rec.*, pages 948–954, 1997.

[36] C. Wren, A. Azarbayejani, T. Darrell, and A. Pentland. Pfinder: Real-time tracking of the human body. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 19(7):780–785, July 1997.

[37] J. Yamato, J. Ohya, and K. Ishii. Recognizing human action in time-sequential images using hidden markov model. *Proc. Comp. Vis. and Pattern Rec.*, pages 379–385, 1992.