Recognition of Visual Activities and Interactions by Stochastic Parsing

Yuri A. Ivanov, *Student Member*, *IEEE Computer Society*, and
Aaron F. Bobick, *Member*, *IEEE Computer Society*

Abstract—This paper describes a probabilistic syntactic approach to the detection and recognition of temporally extended activities and interactions between multiple agents. The fundamental idea is to divide the recognition problem into two levels. The lower level detections are performed using standard independent probabilistic event detectors to propose candidate detections of low-level features. The outputs of these detectors provide the input stream for a stochastic context-free grammar parsing mechanism. The grammar and parser provide longer range temporal constraints, disambiguate uncertain low-level detections, and allow the inclusion of a priori knowledge about the structure of temporal events in a given domain. To achieve such a system we: 1) provide techniques for generating a discrete symbol stream from continuous low-level detectors; 2) extend stochastic context-free parsing to handle uncertainty in the input symbol stream; 3) augment a run-time parsing algorithm to enforce intersymbol constraints such as requiring temporal consistency between primitives; and 4) extend the consistency filtering to maintain consistent multiobject interactions. We develop a real-time system and demonstrate the approach in several experiments on gesture recognition and in video surveillance. In the surveillance application, we show how the system correctly interprets activities of multiple, interacting objects.

Index Terms—Syntactic pattern recognition, action recognition, high level vision, video surveillance, gesture recognition, video monitoring.

1 Introduction

1.1 Structure and Content

In the last several years there has been 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 an 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 from work in gesture recognition [34], [6], [32], where analogies to speech and handwriting recognition have 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.

Our research interests lie in the area of computer vision where observations span extended periods of time and there are particular semantically defined activities we would like to be able to recognize. For example, consider the task of the visual surveillance of a parking lot. We may know a priori that we wish to detect whenever a person is dropped-off or picked-up or if someone drives in but then leaves the scene without first entering a building. These high-level events are composed of primitives (e.g., `car-enter`, `car-stop`, `person-exit`) linked in a spatio-temporal structure that satisfies particular constraints. For example, during a `drop-off`, the `car-stop` event happens near where the corresponding `person-found` detection occurs—where "near" is in both space and time.

Or, consider a simple gesture example—we can draw a square with a hand in the air in either clockwise or counterclockwise direction. In either case, our `square-gesture` recognition system should indicate that the square is being drawn. This seemingly simple task requires significant effort using only the statistical pattern recognition techniques because, although the two squares share primitives, the overall patterns are statistically quite distinct.

In these domains where the goal is to recognize structurally defined relationships of primitives, purely statistical approaches to recognition are less than ideal. These situations can be characterized by one or more of the following properties:

- *Insufficient data*: Complete data sets are not always available, but component examples are easily found;
- *Semantic ambiguity*: Semantically equivalent processes possess radically different statistical properties;

- *Y.A. Ivanov is with the Vision and Modeling Group, MIT Media Laboratory, 20 Ames St., E15-368A, Cambridge, MA 02139. E-mail: yivanov@media.mit.edu.*
- *A.F. Bobick is with the College of Computing, Georgia Institute of Technology, 801 Atlantic Dr., Atlanta, GA 30332. E-mail: afb@cc.gatech.edu.*

Manuscript received 21 Apr. 1999; revised 13 Dec. 1999; accepted 13 Dec. 1999.
Recommended for acceptance by R. Collins.
For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number 109641.

- *Temporal ambiguity*: Competing hypotheses can absorb different lengths of the input stream, raising the need for naturally supported temporal segmentation;
- *Known structure*: Structure of the process is difficult to learn but is explicit and a priori known.

When these conditions arise, it seems natural to divide the problem in two—recognition of primitives and recognition of structure. The goal then becomes to combine statistical detection of primitives with a structural interpretation method that organizes the data.

For many domains such a division is clear. When we speak of many visual activities we often refer to them as naturally segmented sequences of steps, implying some sort of algorithmic structure where primitives are clearly defined. For instance, in the surveillance examples, the (statistical) primitives would be the temporally local behaviors of the vehicles and people; the (structural) patterns are the high-level activities. For a completely different domain, 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 of 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 a `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.

We further motivate this decomposition by noting that the decoupling of the detection of the low-level primitives from the recognition of the higher level activities allows us to create a generic method for the structural analysis. Our only requirement will be that the low-level detectors be able to generate detection events and be able to assign some certainty to characterization of those events. In the results section of this paper, we demonstrate the utility of such a decoupling where, in two recognition systems, we use entirely different techniques of detecting the primitive components while the higher level structure analysis algorithm remains completely unchanged.

In this paper, we propose a method which combines statistical techniques used to detect primitive components of an activity with syntactic recognition of the process' structure. We combine results of the lower level component detectors into a consistent maximally likely interpretation using the Stochastic Context-Free Grammar parser. The grammar provides a convenient means for encoding the external knowledge about the problem domain, expressing the expected structure of the activity.

Syntactic pattern recognition in machine vision has been mostly applied to still images. In those applications, the inherent sequentiality of syntactic parsing mechanisms has been considered a limitation, requiring more sophisticated forms of grammars to be employed. As machine vision advances into the areas of action recognition, sequential machines find progressively more uses. We have already mentioned the growing popularity of Hidden Markov Models (HMM) in computer vision which, in their essence, are probabilistic finite state machines. The relation between Stochastic Context-Free Grammars (SCFG) and HMMs is very similar to that between CFGs and nonprobabilistic Finite State Machines (FSM), where CFGs relax some of the structural limitations imposed by FSMs. In this context, using SCFGs is not a limitation, but, compared to FSMs, a newly rediscovered freedom.

The outline of this paper is as follows: We begin with a review of previous work, including not only work in computer vision on the visual interpretation of activity, but also results in SCFG parsing upon which some of this work is built. We next present the basic SCFG parsing algorithm and our extensions that permit the system to handle the types of ambiguity found in perception problems. We also provide a mechanism for handling streams of input events generated by interacting agents, where not only the behavior of each agent obeys syntactic constraints, but where their interactions do as well. We conclude by demonstrating the utility of the approach on simple but interesting gesture recognition examples and then on a rich visual surveillance task.

2 Related Work

The work we describe in this paper is inspired by research in two main areas—gesture recognition and natural language processing, particularly in the area of stochastic parsing. We attempt to extend these techniques to model more complex structured gestures and activities. A variety of techniques in these areas, which we discuss in the following sections, are closely related to our work.

2.1 HMMs and FSMs in Visual Activity Recognition

Prediction and estimation of a temporally extended time series has been addressed predominantly by state-space models. Of this class of models, HMMs, because of their success in the speech community, have received the most attention in machine vision. Review of such methods as applied to vision can be found in a recent paper by Aggarwal and Cai [1].

One of the earlier attempts to use HMMs for recognition of activities is found in the work by Yamato et al. [40], where discrete HMMs are used to recognize six tennis strokes, performed by three subjects. A 25×25 subsampled camera image is directly used as a feature vector.

A large body of work has arisen from tackling the problem of gesture recognition. The naturally sequential characterization of the task was reflected in the sequential structure of a statistical model, such as work by Darrell and Pentland [16], where the recognition task is performed by a time-warping technique, closely related to HMM methodology. Bobick and Wilson [6] also used dynamic time warping to match an input signal to a deterministic sequence of states.

Examples of statistical representation of sequences are seen in the recent work in understanding human gesture. For instance, Schlenzig et al. [32] describe the results of their

experiments of using HMMs for recognition of continuous gestures, which show to be a powerful gesture recognition technique. Starner et al. [33] 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 related work, Wilson et al. [38] analyzed the explicit structure of the gesture where the structure was implemented by an equivalent of a finite state machine with no learning involved. The distinction there was the use of duration modeling that controlled dwell time in states and the use of features that were relative to the current state of the gesturer, for example, whether the hands were in a "rest" state.

Oliver et al. [25] developed a system for detecting people interactions which modeled interactions using Coupled Hidden Markov Model [9]. In the course of the latter research, a multiagent simulation was used to produce synthetic training data to train the CHMM modeling the interaction. The relation to our work is that their representation of the multiagent simulation can be viewed as a structured, stochastic grammar-like description of the interactions.

State-machine representations of action have also been employed in higher level descriptions. Bremond and Medioni [10] use hand-crafted deterministic automata to recognize airborne surveillance scenarios. Their approach has no mechanism for handling uncertain or incomplete data.

2.2 Stochastic Parsing

The most definitive text on parsing theory still remains [2]. 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 [31].

For efficiency reasons, parsing algorithms often 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 [17]. 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. We mention this work as it was the basis of Stolcke's work (see below), which is an essential precursor to our own approach.

A simple introduction of probabilistic measures into grammars and parsing was shown by Booth and Thompson [7], Thomason [36], and others. The primary interest there is in ambiguous grammars where the likelihood of the derivation can be used to select the appropriate parse.

Aho and Peterson addressed the problem of ill-formedness of the input stream. In [3], 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. Their approach is utilized in this paper in the framework of uncertain input and multivalued strings.

Syntactic approach to multiagent behaviors has been addressed in [15]. Agent interactions are expressed in terms of Parallel Communicating Grammar Systems (PC Systems). The approach presented here extends a single SCFG parser to handle simple concurrency within a single parsing routine.

2.3 Syntactic Techniques in Speech

Probabilistic aspects of syntactic pattern recognition for speech processing were presented in many publications, for instance, in [19], [13]. The latter demonstrates some key approaches to parsing sentences of natural language and shows advantages of use of probabilistic CFGs. The text shows a 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 [35]. 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. Our own work draws heavily on work by Stolcke, as seen in Section 3.

Generally, the model of high-level constraints in speech takes the form of a language model, which acts as a prior probability distribution over sequences of spoken words. Integration of the language model into the optimization routine is straightforward if it is not exceedingly complex. For instance, many algorithms address the case when such a model is represented by a Finite State Network. Such a network corresponds to a language complexity equivalent to the Regular Grammar. For instance, Rabiner and Juang [27] discuss several such models and ways of integrating them with acoustic models. As a more recent reference, a book by Jelinek [21] gives a number of examples of these techniques.

As an example of integration of high- and low-level evidence, one can examine a commercially available system—HMM Tool Kit (HTK) developed by Entropic Research Lab. In HTK, 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 detectors and perform long sequence parse by a Viterbi algorithm based on token passing [41].

The resulting grammar network is shown in Fig. 1a. The structure of the incoming string can be described by a nonrecursive, finite state model.

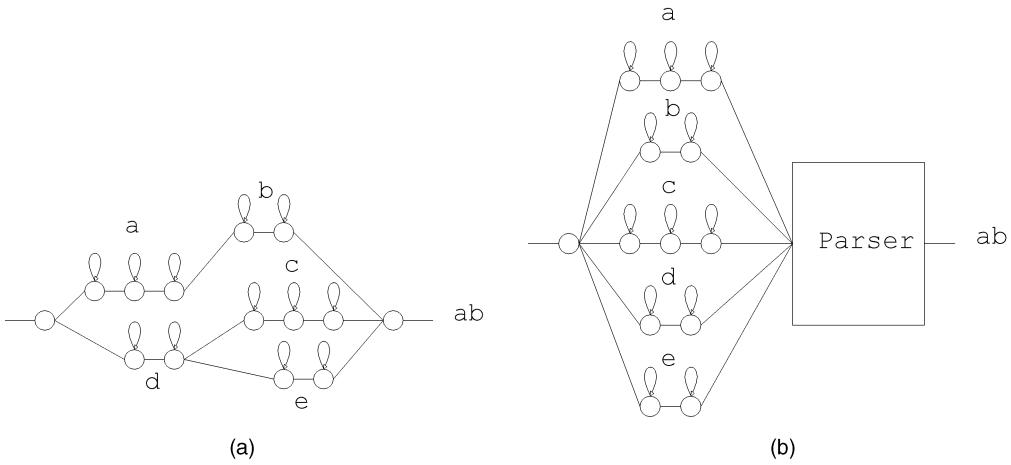


Fig. 1. Illustration of different parsing strategies. (a) Example of HTK—individual temporal feature detectors for symbols a, b,c, d, and e are combined into a grammar network. (b) Proposed architecture which achieves further decoupling between the primitive detectors and the structural model (probabilistic parser in this case).

In contrast, with approaches to integration of high- and low-level evidence in speech, in this paper, we present a method which further decouples high- and low-level models, as shown in Fig. 1b. Such decoupling is desirable in our applications because, in a number of them, low-level events are not always characterized by the form of an object trajectory in some state space, but rather by a momentary measurement, such as its final position. For instance, in the surveillance system demonstrated later in the paper, we only use the end points of object trajectories to represent primitives. Likelihoods of these primitives are simply probabilities associated with object class memberships. Sections 5 and 6 show that clear separation of the machinery of the two levels allows us to use the same high-level model with two different models of low-level components. Potentially, one can imagine using a heterogeneous set of low-level features within the proposed framework.

In addition, speech processing is typically not concerned with detection and synchronization of parallel streams of evidence, which is often a requirement for detecting interactions in vision tasks.

2.4 Syntactic Techniques in Vision

The syntactic approach in Machine Vision has been studied for more than 30 years (e.g., [23], [4]), mostly in the context of pattern recognition in still images. Tsai and Fu [37] employed attributed grammars to combine probability and syntactic constraints in still image understanding. The work by Bunke and Pasche [12] is built upon the previously mentioned development by Aho and Peterson [3], expanding it to multivalued 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 nonprobabilistic context.

More recent work by Sanfeliu and Sainz [30] 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. A more theoretical treatment of the approach is given in [29]. In the latter work, the authors introduce a method of parsing AREs which describe a subclass of context-sensitive languages, including the ones defining planar shapes with symmetry.

An information theoretic approach to stochastic parsing is demonstrated by Oomen and Kashyap in [26]. The authors present a foundational basis for optimal and information theoretic syntactic pattern recognition. They develop a rigorous model for channels which permit arbitrarily distributed substitution, deletion, and insertion syntactic errors. The scheme is shown to be functionally complete and stochastically consistent.

There are several examples of attempts to enforce syntactic and semantic constraints in recognition of visual data. For instance, Courtney [14] uses a graph-based 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." Courtney's approach does not consider probabilistic detections and interpretations, instead relying on perfect low-level sensing.

The grammatical approach to visual activity recognition was used by Brand [8], who used a simple nonprobabilistic 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. Like in Courtney's work, there is no consideration of probability or error.

3 Theoretical Foundations—Stochastic Parsing

This section presents a brief introduction of the stochastic parsing algorithm developed by Stolcke in [35] based upon [17]. In the next section, we extend the components necessary to handle the types of uncertainty encountered in visual recognition.

3.1 Stochastic Context-Free Grammar

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

$$A \rightarrow \lambda[p]. \tag{1}$$

The rule probability p is usually written as $P(A \rightarrow \lambda)$. This probability is a conditional probability of the production being chosen, given that nonterminal A is up for expansion (in generative terms). Saying that a stochastic grammar is context-free essentially means that the rules are conditionally independent and, therefore, the probability of the grammar generating a particular complete derivation is simply the product of the probabilities of rules participating in the derivation.

3.2 Earley-Stolcke Parsing Algorithm

The method most generally and conveniently used in stochastic parsing is based on an Earley parser [17], extended in such a way as to accept probabilities.

In parsing stochastic sentences, we adopt a slightly modified notation of [35]. The notion of a *state* is an important part of the Earley parsing algorithm. A state, S_k^i, is denoted by:

$$i : X_k \rightarrow \lambda.Y\mu \;\; [\alpha, \gamma], \tag{2}$$

where X and Y are nonterminals, λ and μ are substrings, **"."** 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 $w_k \ldots w_i \ldots w_l$, 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 brevity we will denote a state as S_k^i.

For each position of the input stream, the parser keeps a set of states, which denote all pending derivations. After seeding the initial state set with the "dummy" start state, $0 : {}_0\!\rightarrow .Z$, expanding into a topmost symbol of the grammar, Z, parsing proceeds as an iteration of three basic steps—*prediction*, *scanning*, and *completion*. If, after k iterations, the state $k : {}_0\!\rightarrow Z.$ is reached, the corresponding string is accepted. 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 support for further abstraction. A state "explains" a string that it dominates as a possible interpretation of symbols $w_k \ldots w_i$, "suggesting" a possible continuation of the string if the state is not complete.

In a probabilistic parsing algorithm, a state is augmented by two variables which hold forward and inner probabilities, denoted in (2) by α and γ. α, also called a prefix probability, is the probability of the parsed string up to position i and γ is a probability of the substring starting at k and ending at i.

3.2.1 Prediction

In the parsing algorithm, the prediction step is used to hypothesize the possible continuation of the input based on the current position in the parse. Prediction essentially expands one branch of the grammar down to the set of its leftmost leaf nodes to predict the next possible input terminal.

Using the state notation above, for each state $i : X_k \rightarrow \lambda.Y\mu$ and production $Y \rightarrow \nu$, the algorithm produces a new state:

$$\begin{cases} i : X_k \rightarrow \lambda.Y\mu \;\; [\alpha, \gamma] \\ \\ Y \rightarrow \nu, \end{cases} \Rightarrow i : Y \rightarrow .\nu \;\; [\alpha', \gamma'] \tag{3}$$

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:

$$\begin{aligned} \alpha' &= \sum_{\lambda,\mu} \alpha(i : X_k \rightarrow \lambda.Z\mu)\mathbf{R}_L(Z, Y)P(Y \rightarrow \nu) \\ \gamma' &= P(Y \rightarrow \nu). \end{aligned} \tag{4}$$

Matrix R_L is a *Reflexive Transitive Closure* of a *Left Corner Relation* between nonterminals in the grammar. Derivation of the form of the matrix is given in Appendix A. It compensates for recursive relations between the nonterminals in the grammar for probability estimation.

3.2.2 Scanning

Scanning 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 $i + 1 : X_k \rightarrow \lambda a.\mu$ for the next state set:

$$i : X_k \rightarrow \lambda.a\mu \quad [\alpha, \gamma] \Rightarrow i + 1 : X_k \rightarrow \lambda a.\mu \quad [\alpha, \gamma], \tag{5}$$

where α and γ are forward and inner probabilities. 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 (see Section 4.2). Any states that are not confirmed by the scanned input symbol are discarded.

3.2.3 Completion

The completion step, given a set of states which have just been confirmed by scanning, updates marker positions in all pending derivations all the way up the derivation tree. The marker 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 nonterminal. 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:

$$\begin{cases} j : X_k \rightarrow \lambda.Y\mu \ \ [\alpha, \gamma] \\ i : Y_j \rightarrow \nu. \ \ [\alpha'', \gamma'']. \end{cases} \Rightarrow i : X_k \rightarrow \lambda Y.\mu \quad [\alpha', \gamma'] \qquad (6)$$

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 $i : Y_j \rightarrow \nu.$:

$$\begin{aligned} \alpha' &= \sum_{\nu} \alpha(i : X_k \rightarrow \lambda.Z\mu)\mathbf{R}_U(Z, Y)\gamma''(i : Y_j \rightarrow \nu.) \\ \gamma' &= \sum_{\nu} \gamma(i : X_k \rightarrow \lambda.Y\mu)\mathbf{R}_U(Z, Y)\gamma''(i : Y_j \rightarrow \nu.). \end{aligned} \qquad (7)$$

Matrix R_U is a *Reflexive Transitive Closure* of a *Unit Production Relation* between nonterminals in the grammar. Derivation of the form of the matrix is given in Appendix B.

Normally, computations of α and γ are performed incrementally. The closed form (4) and (7) are shown for clarity.

3.2.4 Viterbi Algorithm

After the final state is reached, the sequence of terminals forming the state, if needed, can be recovered by the Viterbi algorithm. It 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:

$$v_i(S_k^i) = \max_{S_j^i}(v_i(S_j^i)v_j(S_k^j)) \qquad (8)$$

and the Viterbi path would include the state:

$$S_k^i = \arg\max_{S_j^i}(v_i(S_j^i)v_j(S_k^j)). \qquad (9)$$

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.

4 Parsing with Perceptual Uncertainty

The SCFG formalism presented so far is primarily concerned with generating and selecting uncertain derivations in response to input streams of completely certain symbols. In our decoupled setting, the source of the input symbols is itself probabilistic. This calls for an extension to the algorithm which allows for inclusion of uncertainties present in input symbols.

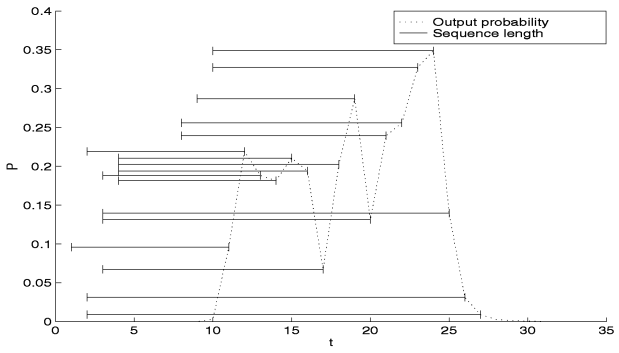


Fig. 2. Output of a component model. Here, at every sample, the activity primitive, modeled by the HMM, outputs a model likelihood. Each point of the probability plot is the normalized maximum likelihood of the HMM at the current sample of the input signal; the maximum is taken over possible starting points prior to the current point in time. The horizontal lines correspond to the temporal interval that yields that maximum value.

4.1 Independent Primitive Detection

Our approach is to combine the detection of independent components of activities in a framework of syntax-driven structure recognition. Typically, these components are detected "after the fact," that is, detection of the primitive only succeeds when the entire component is present in the input signal. At the point when the primitive is detected by the low-level recognizer, we require that the detection be represented by both a *likelihood*, corresponding to the probability that the data observed would be generated by the model of the primitive, and an associated *trace characterization* of the fragment of the input signal associated with this detection. For many situations, the trace is simply the temporal interval spanned by the primitive. We refer to the likelihood as the *terminal likelihood* and to the temporal span of the trace as the *terminal length*.

Fig. 2 illustrates one example of primitive detection. The value of the graph is the maximum normalized output probability of a backward-looking HMM trained to respond to some small sequence of input signal. The maximum is taken over all possible starting points. The horizontal segment extending backward from each point on the graph corresponds to the the interval of input signal that yields the maximum value. If a detection was reported at each time step, the HMM likelihood would be the terminal likelihood and the endpoints of the bar would be the trace. In Section 5, we describe a procedure to convert the temporally continuous output of the HMM into a set of discrete temporal events.

Assumed in our framework is that each detector is working independently, detecting primitives and assigning likelihoods without knowledge of other detections. The task of organizing the best set of detections into a coherent interpretation is accomplished through parsing.

4.2 Uncertainty in the Input

The parsing algorithm, described in Section 3, does not consider uncertainty about the input symbols. This is not very convenient in the proposed architecture since the lower level activity models are typically probabilistic. As suggested in [35], the Earley-Stolcke framework can be extended to incorporate this probabilistic data into the

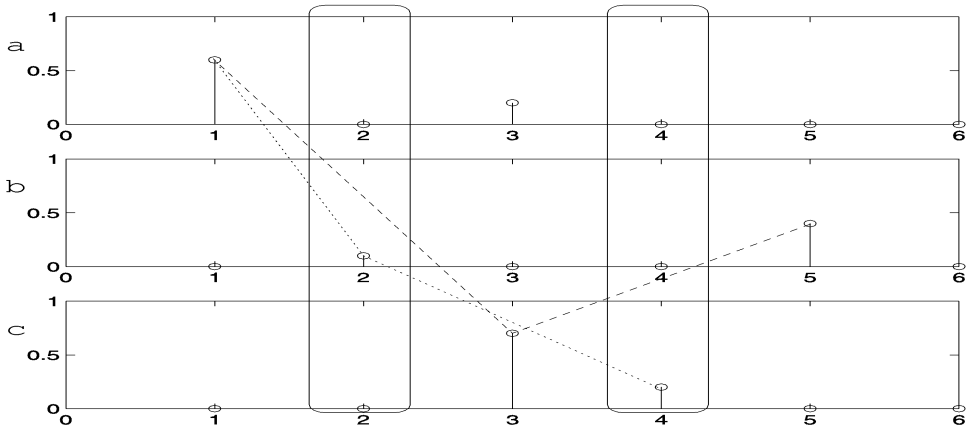


Fig. 3. Example of the lattice parse input for three model vocabulary. Consider a grammar $A \rightarrow abc \mid acb$. The dashed line shows a parse acb. The two rectangles drawn around samples 2 and 4 show the "spurious symbols" for this parse which need to be ignored for the derivation to be contiguous.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

parsing process at the scanning step as follows: First, we allow the scanning step to add several candidate symbols to the chart at each state set. The *likelihood of each input symbol* is incorporated into the parse by multiplying the inner and forward probability of the state being confirmed by the input likelihood.

Taking likelihoods into account the scanning step of (5) is reformulated as follows: For each symbol a with nonzero likelihood $P(a)$ scanning produces the state:

$$\begin{cases} i : X_k \rightarrow \lambda.a\mu \ \ [\alpha, \gamma] \\ \forall a, \quad s.t. P(a) > 0 \end{cases} \Rightarrow i+1 : X_k \rightarrow \lambda a.\mu \ \ [\alpha', \gamma'] \quad (10)$$

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

$$\begin{aligned} \alpha' &= \alpha(i : X_k \rightarrow \lambda.a\mu) P(a) \\ \gamma' &= \gamma(i : X_k \rightarrow \lambda.a\mu) P(a). \end{aligned} \quad (11)$$

The new values of forward and inner probabilities will weigh competing derivations not only by the typicality of corresponding production rules, but also by the certainty about the input at each sample. This technique addresses the correction of substitution errors by considering several candidate inputs at each time step and only committing to the particular value once the whole sequence has been observed.

The input stream can now be viewed as a *multivalued string* (a lattice) which has a vector of likelihoods, associated with each time step. In Fig. 3, at each time step (labeled 1 to 5), a vector of likelihoods is generated where the likelihood of each individual symbol (a, b, or c) is indicated by the height of the spike. For example, at time step 2, only terminal b has any nonnegligible likelihood, while, at time step 3, both a and c are possible.

4.3 Insertion

Suppose that the recognition of the primitives of the vocabulary is performed by a set of independent models. This implies that the detections are not mutually exclusive. At each such detection, the parser needs to select the one which results in the highest probability parse. This issue has been only partially addressed by considering substitution errors. Indeed, the primitives are detected asynchronously, which results in the appearance of *insertion* errors.

On one hand, the erroneous symbols have to be removed from the stream, but, on the other, we need to preserve each such symbol in the stream for considering it in other possible derivations, perhaps even of a completely different string.

In our applications (recognition of structured visual activities), there are two primary issues which dictate the solution to this problem:

1. *Misdetections*. Since the models are treated independently, they do not suppress one another, producing erroneous symbols. However, until the whole sequence is observed, no decision can be made about whether these symbols are actual detections or noise, as illustrated in Fig. 3.
2. *Concurrent activities*. A more interesting issue is related to the possibility of using the same parsing process to detect and label concurrent independent activities. Suppose your input consists of events $abcd$. In this input, events a and c belong to one entity, while events b and d the other. The parser can produce correct parses if it is capable of correcting insertion errors.

To address correction of insertion (and deletion) errors, the activity grammar is converted to "robust" form which, in addition to correcting errors, allows the inclusion of additional constraints on the input stream easily, as will be shown in later sections.

The robust form $\hat{G}$ of a grammar G is formed by the following rules:

1. Each terminal, say b, appearing in productions of grammar G is replaced by a preterminal, e.g., $\hat{B}$, in $\hat{G}$:

$$\begin{array}{lcl} G: & \Rightarrow & \hat{G}: \\ A \rightarrow bC & & A \rightarrow \hat{B}C. \end{array}$$

2. For each preterminal of $\hat{G}$, a `SKIP` rule is formed:

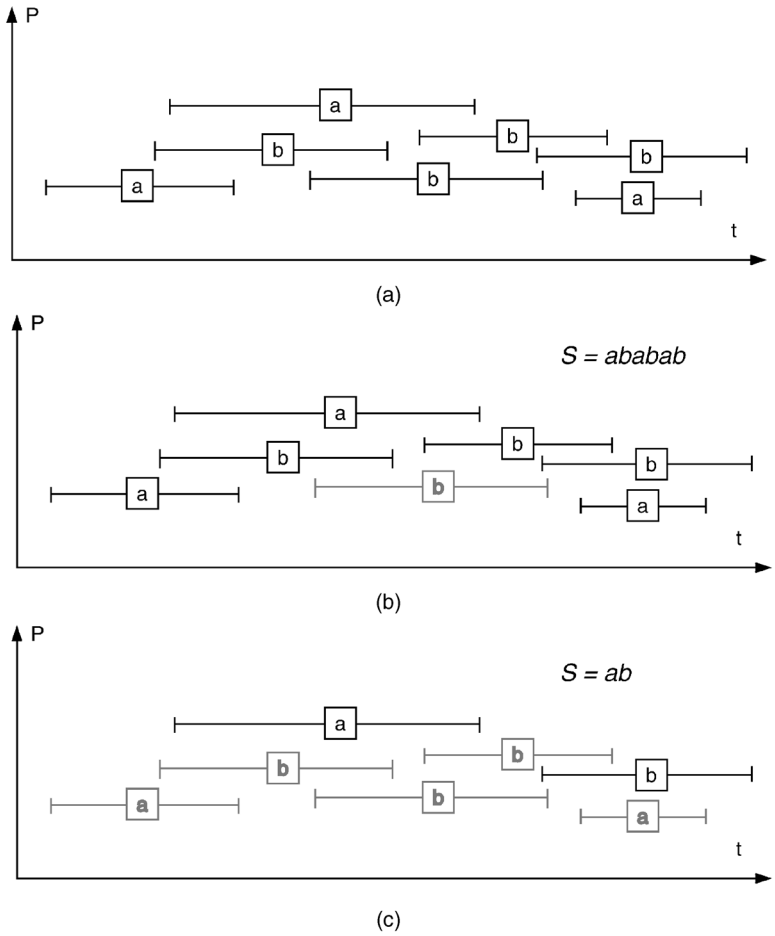


Fig. 4. Example of temporal consistency. Data in (a) is produced by two component models—a and b. Recall that the right edge of each bar designates the detection time (refer to Fig. 2). Given a production, $A \rightarrow ab \mid abA$, an unconstrained parse will attempt to consume maximum amount of samples by non-$SKIP$ productions. The resulting parse, $ababab$, is shown in (b), where rejected symbols are shown in gray. (c) Shows a temporally consistent parse, ab, which includes only nonoverlapping terminals.

$$\begin{array}{l}\hat{G}: \\ \hat{B} \quad \rightarrow b \mid SKIP \;\; b \mid \;\; b \;\; SKIP.\end{array}$$

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

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

$$\begin{array}{l}\hat{G}: \\ SKIP \rightarrow b \quad\quad \mid c \mid \quad\quad \ldots \\ \mid \; b \;\; SKIP \mid \quad\quad c \;\; SKIP \mid \quad\quad \ldots.\end{array}$$

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

$$\begin{array}{l}\hat{G}: \\ \hat{B} \quad\quad \rightarrow \quad\quad SKIP \;\; b \;\; SKIP \\ SKIP \;\; \rightarrow \;\; \epsilon \mid \;\; b \;\; SKIP \;\; \ldots.\end{array}$$

This conversion can be performed automatically as a preprocessing step when the grammar is read in by the parser so that no modifications to the grammar are explicitly written.

It is easy to see that the robust grammar will consume the erroneous symbols by the $SKIP$ production. However, we now have the additional concern of the value of probability of the $SKIP$ production. In our experiments, we set it to a relatively low value to penalize derivations that do not consume the maximum number of terminals.

4.4 Enforcing Internal Consistency

As noted earlier, the low-level primitive detections each correspond to a particular temporal span of the input signal; also, properties of the trace of the signal are associated with the detection. If these data are available to the parser, they can be easily used as additional constraints on the input terminals. The goal of the parsing is to extract a coherent single-stream sequence of primitives (observations). These primitives should be required to be consistent and label nonoverlapping parts of the input signal. This point is further illustrated in Fig. 4. If terminal length is not considered, the data of Fig. 4a is parsed as $ababab$, as shown in Fig. 4b, where only one terminal b is discarded. The correct consistent parse shown in Fig. 4c, where no overlapping terminals are accepted.

In the Earley framework, the completion step presents a good opportunity to implement a general mechanism of enforcing terminal consistency. When considering a pair of states for joining at completion, the parser can reject or penalize inconsistent states. The completion step would

work as an internal consistency filter, for example, keeping track of the terminal lengths during scanning and prediction.

To accomplish this, two additional state variables are introduced—h for "high mark" and l for "low mark." These variables, in general vector-valued, contain all the data that is available and necessary for the parser to compute the distance penalty, that is, the penalty for asserting that one particular instance of some terminal follows a particular instance of another terminal. This mechanism is fairly general and concrete examples of usage will be given in the experimental section.

Each of the parsing steps is properly adjusted to maintain the low and high marks:

1. **Prediction**
   Prediction simply marks the expected beginning of the substring with the initial values S_t:

$$\begin{cases} i : X_k \rightarrow \lambda.Y\mu \ \ [\mathbf{l}, \mathbf{h}] \\ \\ Y \rightarrow \nu. \end{cases} \Rightarrow i : Y \rightarrow .\nu \ \ [\mathbf{S}_t, \mathbf{S}_t] \quad (12)$$

2. **Scanning**
   Scanning reads a terminal from the input stream and sets high marks of scanned states to the high mark of the terminal, expanding the range of the state. In addition, for all the *predicted* states expecting this terminal, it sets the low mark to the low mark of the scanned terminal:

$$\begin{array}{l} i : X_k \rightarrow \lambda.a\mu \ \ [\mathbf{l}, \mathbf{h}] \Rightarrow \\ \begin{cases} \quad i+1 : X_k \rightarrow \lambda a.\mu \ \ [\mathbf{l}_a, \mathbf{h}_a], & if \lambda = \varepsilon \\ i+1 : X_k \rightarrow \lambda a.\mu \ \ [\mathbf{l}, \mathbf{h}_a], & otherwise, \end{cases} \end{array} \quad (13)$$

   where $\mathbf{l}_a$ and $\mathbf{h}_a$ are low and high mark attributes of the terminal, respectively.
3. **Completion**
   The completion step advances the high mark of the completed state to that of the completing state, thereby extending the range of the completed nonterminal:

$$\begin{cases} j : X_k \rightarrow \lambda.Y\mu \ \ [\mathbf{l}_1, \mathbf{h}_1] \\ \\ i : Y_j \rightarrow \nu. \ \ [\mathbf{l}_2, \mathbf{h}_2]. \end{cases} \Rightarrow i : X_k \rightarrow \lambda Y.\mu \ \ [\mathbf{l}_1, \mathbf{h}_2] \quad (14)$$

   The completion is performed for all *complete* states $i : Y_j \rightarrow \nu.$, subject to consistency constraints enforced within the filtering routine.

4.4.1 Consistency Filtering

The consistency filtering routine is invoked from the completion step. It is the routine which determines how to handle the two states which are considered for joining. The filtering routine computes the distance, d, between two states based on $\mathbf{l}$ and $\mathbf{h}$ marks of the candidate states. Based on this distance, the penalty according to which forward and inner probabilities are computed. Assuming that the penalty function $f(d)$ has the range $[0; 1]$, update equations for α and γ can be written as follows:

$$\begin{array}{l} \alpha' = f(d) \sum\limits_{\forall \lambda, \mu} \alpha(i : X_k \rightarrow \lambda.Z\mu)\mathbf{R}_U(Z, Y)\gamma''(i : Y_j \rightarrow \nu.) \\ \gamma' = f(d) \sum\limits_{\forall \lambda, \mu} \gamma(i : X_k \rightarrow \lambda.Y\mu)\mathbf{R}_U(Z, Y)\gamma''(i : Y_j \rightarrow \nu.). \end{array} \quad (15)$$

In its simplest form, the function $f(d)$ can be a step function $f(d) = \Theta(d)$. This choice of the penalty function results in the absolute rejection of any states that violate the internal consistency constraints. If the distance d is only based upon the temporal endpoints of the states, then such a rejection function eliminates interpretations in which any components overlap in time.

An alternative, softer penalty can result in different behavior. For instance, $f(d) = Ce^{-\frac{d^2}{\theta}}$, where C is a normalizing constant and θ, a penalizing parameter, can be used to weigh "overlap" and "spread" equally. Specific examples of $f(d)$ will be given in the experimental section.

4.5 Run-Time Incremental Parsing

At run time, the parser is presented with a potentially infinite input stream. It limits the computational complexity ($O(n^3)$) by pruning the states which have probabilities falling below a certain limit and only keeping a parsing chart of a fixed but sufficient length. 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.

These observations call for the following modifications to the algorithm, which make run-time computations possible:

1. A robust grammar can now 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 new state set should be seeded with a "dummy" starting state, $k :_k \rightarrow .Z$, where Z is the topmost nonterminal of the grammar. 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 [16]. The exception is that no "backward" training is necessary since we have an opportunity to "seed" the state set with an axiom at an arbitrary position.

Furthermore, the parser chart does not need to be reparsed for each sample. In fact, the algorithm can be conveniently sped up by performing the parse incrementally. At every step, the current state of the parser encodes all the history by the seed states within the window. The task is now to just perform the next iteration with the new

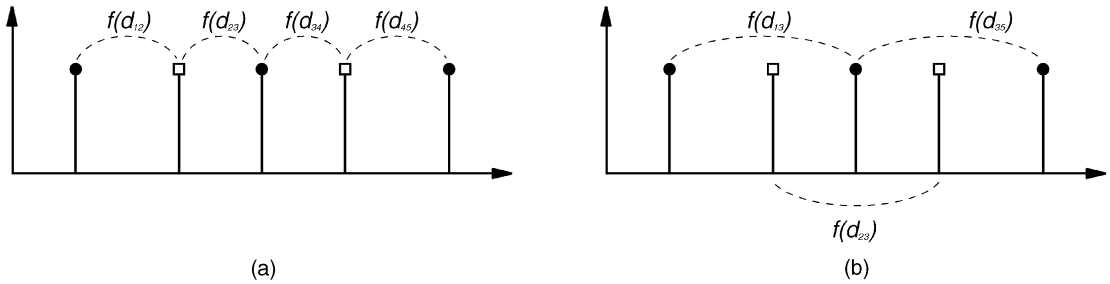


Fig. 5. Performing an interleaved consistency check on the serialized events. Class membership of each event in the plot is designated by a circle or a square at the top of each sample. (a) Serialized events are penalized by the function $f(d)$ computed on consecutive events. This is clearly an incorrect measure. Interleaved consistency check in (b) shows a correct assignment of samples for the penalty function, where the predecessor of the same class is found earlier in the stream and consistency with the predecessor is enforced.

sample, discarding the first state set of the chart. This procedure effectively prunes derivations, which are longer than the length of the chosen window. This also means that an additional condition has to be added to the filtering routine—all the states S_k^i, having $k < I_1$, where I_1 is the index of the first state set of the chart, are rejected from completion.[1]

4.6 Concurrency

Until now, we have presumed that, while there may be spurious and incorrect symbols in the input stream, the goal is to recover a single-stream interpretation. However, actions and activities of interest often include multiobject interactions. Here, we extend the approach to handle the situation in which concurrent activities occur and where interaction between the primitives of different objects is required to instantiate a particular interpretation.

In the approach presented so far, there are three sources of concurrency for which the parser has to account:

1. *concurrent detections*, which are due to probabilistic nature of the low-level primitive detectors,
2. *concurrent parses*, which occur while tracing the derivations of unrelated objects,
3. *concurrent primitives*, which are a part of an interaction between multiple objects.

The parser extended to handle uncertainty in the input stream already handles the first two points. Indeed, *concurrent detections*, being simply the substitution errors, are handled in the usual manner as a multivalued input string. *Concurrent parses* are traced as an added benefit of the error correction with the robust grammar [3].

New to our framework are *concurrent tracks*; these occur in the derivation when the grammar describes interaction between two or more objects; for instance, these could be people and vehicles in the parking lot surveillance applications. The difficulty here is presented by the fact that the primitives in the input stream correspond to different entities. The entities should have internally consistent terminals, but since the events related to both objects in the input stream are interleaved, the consistency check should account for that also. To accomplish that, we modify the parser in three ways:

1. Assign a class label to each production rule of the grammar. For example, in the parking lot monitoring task (Section 6), productions related to cars will have a car class label.[2]
2. For generic consistency, implement a simple search in the filtering routine (Section 4.4.1), which would search the state being completed for the last child state having the class attribute the same as the completing state. The high mark is extracted from the child state and the penalty function is computed based on that attribute, instead of the high mark of the state itself. Fig. 5 shows how the interleaved consistency check is performed on a string of the serialized primitives of two different classes.
3. To enforce interobject consistency, construct specialized distance functions for filtering the combinations of particular pairs of types of parsing states. For example, if to see a `PICK-UP`, the `CAR-STOP` position (as contained in its consistency parameter **h**) needs to be close to the `PERSON-LOST` position (maintained in its **l** parameter), then any completion which joins those two symbols needs to use the appropriate distance function.

5 Experiments—Structured Gesture

In this section, we introduce two gesture recognition experiments, where recognition is performed by the system. In these experiments, the system performs recognition of structured single-stream gestures. The primitives of these complex gestures are simple hand trajectories, which are customarily modeled by Hidden Markov Models (HMMs). To model the gesture, we describe its form as a Stochastic Context-Free Grammar, where terminals of the alphabet correspond to HMMs in the HMM bank. Each of the HMMs picks out a part of the trajectory which is the most similar to the primitive gesture on which it has been trained, estimating the likelihood of the corresponding model. The outputs of the HMMs are then mapped onto a set of discrete events, which correspond to peaks in the HMM outputs and

1. Note that the length of the chart is indexed by events, not time. The value of the pruning threshold and the chart size are chosen such that cutting the chart almost never prunes active states.

2. The rule class should not be confused with the object class. The rule class is not necessarily related to the object class, although in our case it is. For instance, if one needs to handle person-person interactions with the same mechanism, different class labels should be attached to the rules corresponding to each participant.

are passed to the parser, which attempts to find the most likely interpretation of the event set.

5.1 Recognition System

The recognition system for these experiments runs on an R4400 200MHz SGI Indy, performing offline processing of the data collected beforehand using the stereo vision system.

5.1.1 Primitive Recognition

To recognize the components of the model vocabulary, we train one HMM per atomic gesture. At run-time, each of these HMMs performs a Viterbi parse [27] of the incoming signal and computes the likelihood of the corresponding gesture primitive. The run-time algorithm used by the HMMs to recognize the words of the gesture vocabulary is a version of [16] which performs a "backward" match of the signal over a window of a reasonable size. At each time step, the algorithm outputs the estimated likelihood of the sequence which ends at the current sample, as well as the length of this sequence. We will later exploit this property to enforce *temporal consistency*: Only one low-level temporal feature is happening at any one time.

All the HMMs are run in parallel, providing the parser with maximum normalized probability and the corresponding sequence length at each time step.[3] Fig. 2 shows the output of a single HMM in a bank, illustrating the relation between output probabilities and sequence lengths.

5.1.2 Event Generation

The continuous vector output of the HMM bank needs to be mapped onto a discrete alphabet of the grammar, which implies the necessity of converting the output to a series of "events" which will be considered for probabilistic parsing. At this step, no strong decisions about discarding any events need to be made. The parser just needs a sufficient stream of such events so that it can perform structural rectification of these candidate events and find a plausible interpretation which is structurally consistent (i.e., grammatical), is temporally consistent (uses nonoverlapping sequence of primitives), and which has maximum likelihood given the grammar and the outputs of the low-level feature detectors (HMMs).

For the tasks discussed here, a simple discretization procedure provides good results. For each HMM in the bank, a very small threshold is selected to cut off the noise in the output and then a search for a maximum is performed in each area of nonzero probability. Fig. 9a shows an example of the output of the HMM bank, superimposed with the results of discretization.

After discretization, the continuous time vector output of the HMM bank is replaced with the discrete symbol stream generated by discarding any interval of the discretized signal in which all values are zero. Each event in this stream is time stamped, so the time scale is not lost. Fig. 9b displays an example of the resulting event sequence generated.

3. Alternatively, we can search for the position in the trellis 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.

5.1.3 Parser

For this system, the consistency is fully captured by the timing of the signal. Consequently, we chose the consistency parameters $\mathbf{l}$ and $\mathbf{h}$ (Section 4.4) to simply be the time stamps of the beginning and the end of the corresponding primitive

$$\mathbf{l} = t_l \qquad \mathbf{h} = t_h. \tag{16}$$

Confirming consistency, the completion step will now assure that the candidate parses only contain nonoverlapping primitives. This is easily accomplished by choosing the step function, $\Theta(\mathbf{l}_2 - \mathbf{h}_1)$, as the penalty function $f(d)$ (15):

$$f(d) = \begin{cases} 0, & if\ d < 0 \\ 1, & otherwise, \end{cases} \tag{17}$$

where $d = \mathbf{l}_2 - \mathbf{h}_1$. With this choice of the penalty function, overlapping primitives are simply rejected.

5.2 Disambiguation by Context

In this experiment, we address recognition of a simple structured gesture which can take several possible forms. We define a `SQUARE` gesture (Fig. 6) as either lefthanded (counterclockwise) or a righthanded (clockwise) gesture which consists of four parts—`TOP`, `BOTTOM`, `LEFT-SIDE`, and `RIGHT-SIDE`. In this formulation, `TOP` and `BOTTOM`, for example, are ambiguous because both of them can be formed by the same gesture. We note, however, that it can never happen in the same context. That is, if it is a righthanded square, `TOP` is a left-to-right movement and `BOTTOM` is a right-to-left one. In the case of the lefthanded square, the definitions are reversed. We attempt to semantically disambiguate these definitions and recognize a `SQUARE` regardless of the fact that it can be either the righthanded or lefthanded.

To describe this structure, we use a grammar G_{square}: which reflects the ambiguity of the terminal meaning, with `SKIP` rules omitted for simplicity:

G_{square} :

`SQUARE`	$\rightarrow$	`RH`	[0.5]
	|	`LH`	[0.5]
`RH`	$\rightarrow$	`TOP up-down BOT down-up`	[1.0]
`LH`	$\rightarrow$	`BOT down-up TOP up-down`	[1.0]
`TOP`	$\rightarrow$	`left-right`	[0.5]
	|	`right-left`	[0.5]
`BOT`	$\rightarrow$	`right-left`	[0.5]
	|	`left-right`	[0.5].

The trajectory data is collected from a STIVE vision system [39], shown in Fig. 8. The system uses stereo to determine x-y-z position of person's hands and head. At a frame rate of approximately 20 frames per second, the `SQUARE` data set consists of 150-200 samples for each experiment.

For terminal recognition, four three-state HMMs are trained on x and y velocities of 20 examples of each of the primitive hand movements. After achieving reasonable recognition rate, we performed several `SQUARE` gestures determining candidate events as peaks in the likelihood. The results were passed to the parser yielding the output, presented in Fig. 7. Fig. 7a shows that the semantic structure

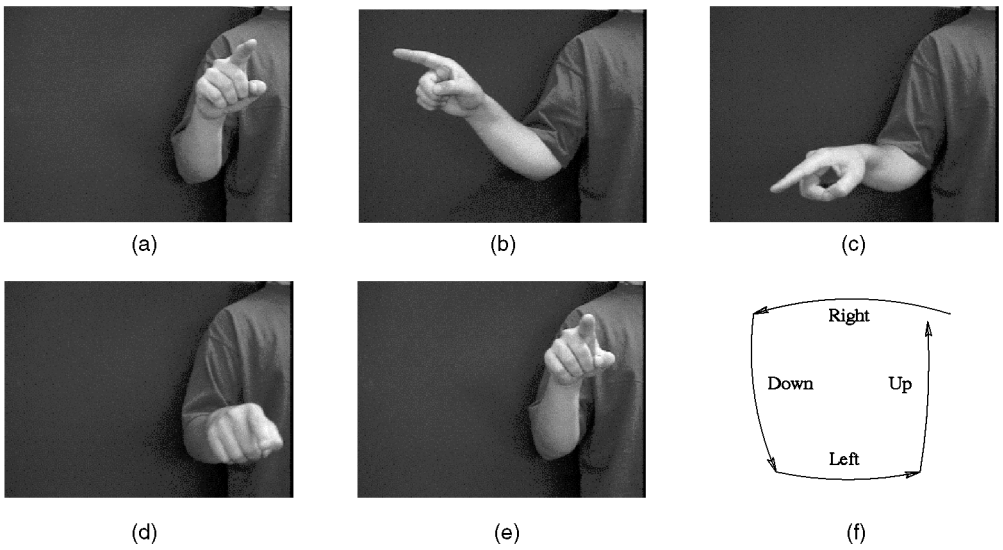


Fig. 6. Example of the activity structure. (a)-(e) Frames of a video sequence showing a $SQUARE$ gesture. (f) Shows the motion phases.

```
Segmentation <rsquare.dat>:
Label              Segment
---------------------------
Right hand square [0  146]
        Top       [0  23 ]
        Up down   [23 66 ]
        Bottom    [66 94 ]
        Down up   [94 146]
Viterbi probability =  0.02400375
```

(a)

```
Segmentation <lsquare.dat>:
Label              Segment
---------------------------
Left hand square  [0   173]
        Bottom    [0   49 ]
        Down up   [49  71 ]
        Top       [71  132]
        Up down   [132 173]
Viterbi probability =  0.01651770
```

(b)

Fig. 7. SQUARE sequence segmentation. (a) Righthanded square and (b) lefthanded square.

recovered was that of a righthand square and the whole sequence was labeled as a SQUARE. Recognition results for a lefthanded square sequence are shown in Fig. 7b. Note that the left-right gesture was interpreted as TOP in the global context in the first case and as BOTTOM in the second. The figures show that timing constraints propagated through the parse and formed consistent continuous coverage of the input signal.

5.3 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 [28] is called "The Grammar of Conducting"). We capture the data from a person who is a trained conductor and who uses natural conducting gestures. The problem we are attempting to solve is the following. A piece of music by Sibelius[4] 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. Both 2/4 and 3/4 gestures are dramatically different from the original 6/4 one, but groups of them coincide with the original 6/4 at bar boundaries. For the experiment, we collect the data from a conductor conducting a few bars of the score,

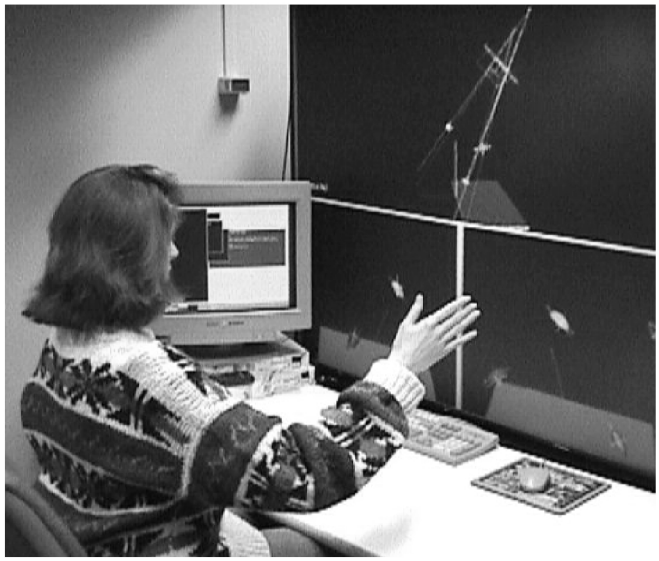


Fig. 8. The Stereo Interactive Virtual Environment (STIVE) computer vision system used to collect data.

4. Jean Sibelius (1865-1957), Second Symphony, Opus 43, in D Major.

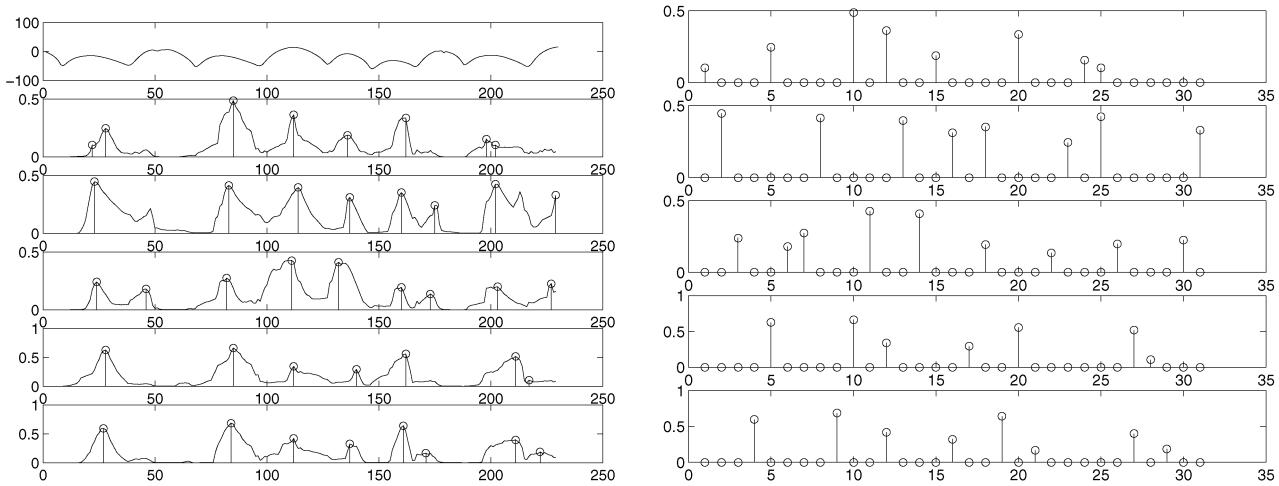


Fig. 9. Output of the HMM bank. (a) Top plot shows the vertical position of the hand throughout the sequence. The five plots below it show output of each HMM in the bank superimposed with the result of discretization. (b) Corresponding discretized "compressed" sequence. Input vectors for the probabilistic parser are formed by taking the samples of all sequences for each time step.

arbitrarily choosing `2/4` or `3/4` gestures, and attempt to find bar segmentation, simultaneously identifying the beat pattern used.

Experimental setup is essentially the same as in the previous example. We trained five HMMs on two primitive components of a `2/4` pattern and the three components of a `3/4` pattern. Some of the primitives in the set are very similar to each other and, therefore, corresponding HMMs show high likelihood at about the same time, which results in a very "noisy" lattice (Fig. 9a). We parse the lattice with the grammar G_c (again, for simplicity omitting the `SKIP` productions):

G_c :

PIECE	→	BAR PIECE	[0.5]
	|	BAR	[0.5]
BAR	→	TWO	[0.5]
	|	THREE	[0.5]
THREE	→	down3 right3 up3	[1.0]
TWO	→	down2 up2	[1.0].

The results of a run of the lower level detectors on a conducting sequence `2/4-2/4-3/4-2/4` are shown in Fig. 9 with the top plot of Fig. 9a displaying a y positional component. Fig. 10 demonstrates output of the parsing algorithm. The output is shown as a set of semantic labels and corresponding sample index ranges, showing a great deal of semantic filtering, where `SKIP` states account for large portion of the input samples.

6 Surveillance

The approach we have presented is suitable for real-time video processing. We present a system built for an online parking lot monitoring task for automatic video surveillance. In this application, our focus is on implementing a technique for automatic detecting of multiobject interactions. Such interactions include simple sequential interactions, such as people driving into the parking lot and leaving the parking lot on foot, as well as more complex ones of people being dropped off or picked up by moving vehicles. The system is capable of handling simple interactions by means of enforcing interleaved consistency between temporally extended primitives. We presently describe the recognition system.

6.1 Recognition System

The recognition system consists of three main components: a tracker, a tracking event generator, and a parser. The system, in general, follows the same two-level architecture as the gesture recognition system of the previous section. The lower level detections in this application represent independent object tracks, which are then mapped onto a set of discrete events. These events form the basis of the parser vocabulary and are processed according to the activity grammar.

6.1.1 Tracker

The tracker used in the recognition system is fully described in [18]. The tracker implements an adaptive background

```
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
```

Fig. 10. Results of the segmentation of a long conducting gesture for the bar sequence `2/4-2/4-3/4-2/4`.

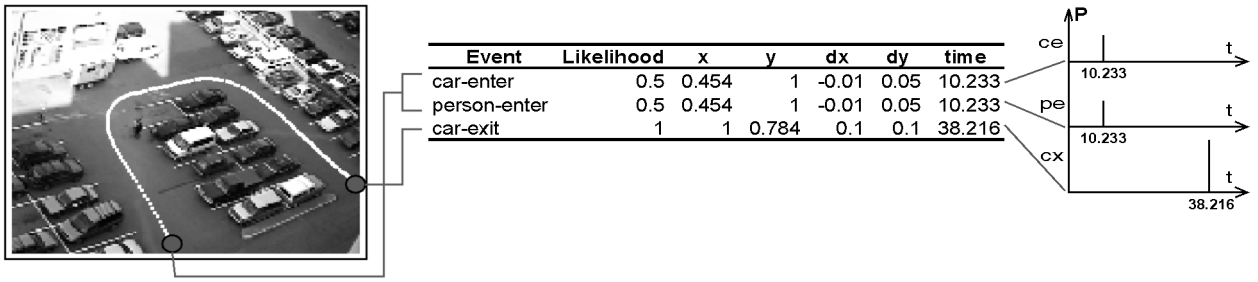


Fig. 11. Illustration of the process of mapping tracks onto discrete events. The tracker reports beginning and the end of the track. In this example, the beginning of the track corresponds to an object entering the scene. At that point, the class label of the object cannot be determined. This results in the generation of two concurrent events—one per object class (cars and persons) with probability of the label being 0.5.

G_p :

```
TRACK          →    CAR-TRACK                                          [0.5]
                  |  PERSON-TRACK                                      [0.5]
CAR-TRACK      →    CAR-THROUGH                                        [0.25]
                  |  CAR-PICKUP                                       [0.25]
                  |  CAR-OUT                                          [0.25]
                  |  CAR-DROP                                         [0.25]
CAR-PICKUP     →    ENTER-CAR-B CAR-STOP PERSON-LOST B-CAR-EXIT        [1.0]
ENTER-CAR-B    →    CAR-ENTER                                          [0.5]
                  |  CAR-ENTER CAR-HIDDEN                             [0.5]
CAR-HIDDEN     →    CAR-LOST CAR-FOUND                                 [0.5]
                  |  CAR-LOST CAR-FOUND CAR-HIDDEN                    [0.5]
B-CAR-EXIT     →    CAR-EXIT                                           [0.5]
                  |  CAR-HIDDEN CAR-EXIT                              [0.5]
CAR-EXIT       →    car-exit                                           [0.7]
                  |  SKIP car-exit                                    [0.3]
CAR-LOST       →    car-lost                                           [0.7]
                  |  SKIP car-lost                                    [0.3]
CAR-STOP       →    car-stop                                           [0.7]
                  |  SKIP car-stop                                    [0.3]
PERSON-LOST    →    person-lost                                        [0.7]
                  |  SKIP person-lost                                 [0.3]
```

Fig. 12. A `CAR-PICKUP` branch of a simplified grammar describing interactions in a parking lot.

model, which is tolerant to slow lighting changes. The tracker detects objects in the scene by the presence of motion in the camera view. If this motion persists for more than some small predetermined number of frames, the object is detected and a new track container is created for it to accumulate the tracking data related to this object. The tracker assigns a unique ID to each newly found object and tracks changes in the object's appearance, position, and velocity, reporting them to the tracking event generator. Based on appearance and trajectory properties, the tracker probabilistically assigns to each object a class label (a car or a person).

The primary difference between this system and the gesture example is that, for the gesture primitives, the actual motion of the object (hand) provided strong indication of primitive identity. In this surveillance application, the exact trajectories are not as important and play no role in the labeling of the primitives. However, the trajectory data are passed to the parser and are available within the general consistency mechanism and can be easily used for computing the distance parameter of the penalty function.

6.1.2 Tracking Event Generator

We formulate interactions between objects in terms of *tracker states*, rather than object trajectories, as described below. The tracker can "lose" an object and then "find" it again later, but it need not reason about the identity of the newly found object. This set of primitive tracker states, such as `object-lost`, `object-found`, forms the alphabet of the interaction presented to the parser in form of a grammar. In this formulation, preserving the identity of an object throughout the scene is not important to the parser. The identity is preserved by the tracker where it is simple for it to do so, such as inside the consistent tracks. Then, the task of associating the disjoint pieces of tracks falls onto the parser.

The event generator performs a mapping of the tracks onto a set of events, which are passed to the parser. The events are generated based on a simple environment model in the following way:

Event	UID	Avg. Size	Class	P	x	y	t	frame
ENTER	38	0.071757	0	0.5	0.958069	0.700315	921206196.9	105
STOPPED	38	0.071757	0	0.994145	0.741973	0.410192	921206203.6	153
ENTER	58	0.027833	1	0.5	0.309444	0.206612	921206203.5	152
PERSON-LOST	58	0.029143	1	0.999875	0.697794	0.3131	921206211.3	207
CAR-LOST	38	0.074138	0	0.992876	0.734908	0.378986	921206214.3	227
FOUND	116	0.088693	0	0.5	0.738265	0.397251	921206219	261
CAR-LOST	116	0.091532	0	0.990221	0.653375	0.333216	921206221.1	276
FOUND	136	0.093981	0	0.5	0.577869	0.288348	921206221.6	279
CAR-LEAVE	136	0.096201	0	0.989932	0.54037	0.991736	921206230.9	343

Fig. 13. A sequence of events corresponding to PICKUP, shown in Fig. 14. The car was lost twice after picking up the person.

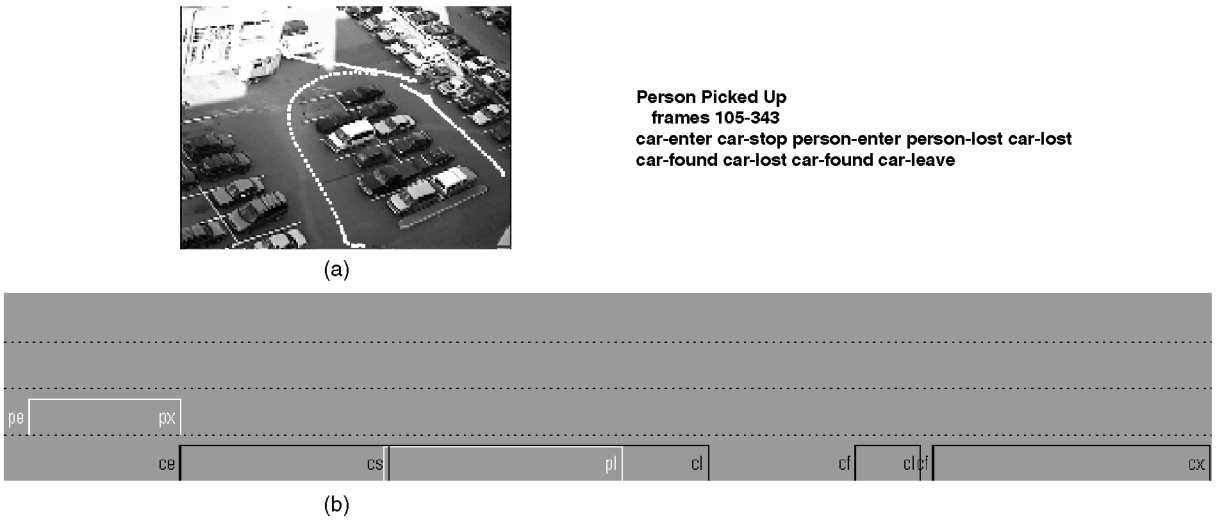


Fig. 14. (a) Shows the iconic representation of PICKUP and the resulting interpretation, generated by the parser. The reassembled tracks are shown in the picture by white traces and correspond to a car and a person. Breaks in the track can clearly be seen. (b) Shows the temporal extent of the action. Actions related to people are shown in white. Car primitives are drawn in black. In this figure, primitive events are abbreviated as follows: ce—car-enter, cs—car-stop, cx—car-exit, pe—person-enter, pl—person-lost, and px—person-exit. The figure shows the concurrency of the person and vehicle tracks and the fact that they are internally consistent.

1. If the track began in an area where objects tend to enter the scene, car-enter and person-enter events are generated.
2. If the track did not begin in one of the "entry" areas, car-found and person-found events are generated.
3. If the track ended in one of the "exit" areas, car-exit and person-exit events are produced.
4. If the track did not end in one of the "exit" areas, car-lost and person-lost events are posted.
5. If an object's velocity dropped below a certain threshold, an object-stopped event is generated.

The process of generating events is illustrated in Fig. 11. All events are marked with corresponding likelihoods. To account for errors in classification, events related to object type always simultaneously generate events related to the other object type. For instance, if a beginning of a person track is reported by the tracker and the likelihood of that event is 0.7, a person-enter event with likelihood 0.7 is posted to the parser. Along with it, a complementary event car-enter is posted in the same time slot with the likelihood of 0.3. Such coupling allows the parser to treat the classification errors introduced by the tracker as substitution errors and treat them syntactically— the parser will select one or the other, depending on which one results in the overall parse with maximum probability. Typically, at the beginning of each track, the tracker has not observed the object long enough to be certain about its class membership. Therefore, X-enter and X-found events have likelihoods close to 0.5. In contrast, by the time the object disappears or is lost, there is enough data to make more accurate classification decision. Consequently, class likelihoods of X-exit and X-lost events are typically more committal than those of X-enter and X-found.

6.1.3 Parser

In accordance with the general consistency filtering mechanism (Section 4.4.1), consistency parameters of the tracks and the form of the penalty function need to be chosen. In the case of surveillance application, actual spatio-temporal distance between endpoints of object tracks can be computed for which the parser needs object position, velocity, and timing data. These data are readily available from the tracker. Then, we chose $\mathbf{l}$ and $\mathbf{h}$ to be:

$$\mathbf{l} = \begin{pmatrix} f_l \\ t_l \\ x_l \\ y_l \\ dx_l \\ dy_l \end{pmatrix} \qquad \mathbf{h} = \begin{pmatrix} f_h \\ t_h \\ x_h \\ y_h \\ dx_h \\ dy_h \end{pmatrix}, \tag{18}$$

	Event	UID	Avg. Size	Class	P	x	y	t	frame	
Person 1	ENTER	3526	0.028702	1	0.5	0.970261	0.663102	921204143.9	955	
	STOPPED	3543		0	0	0.081111	0.150413	921204154.9	1031	
	PERSON-LOST	3526	0.030199	1	0.999873	0.403637	0.573237	921204156.6	1043	
	ENTER	3557	0.038266	1	0.5	0.434028	0.980888	921204157.2	1047	Person 2
	FOUND	3556	0.030252	1	0.5	0.383918	0.569378	921204157	1046	
	PERSON-LEAVE	3556	0.03178	1	0.999999	0.007068	0.289859	921204166.1	1108	
	PERSON-LEAVE	3557	0.034484	1	0.999994	0.321716	0.029368	921204182.6	1224	

Fig. 15. Events generated for Fig. 16. Two people take part in the sequence of the evnts shown here, as can be seen from the UID column of the table.

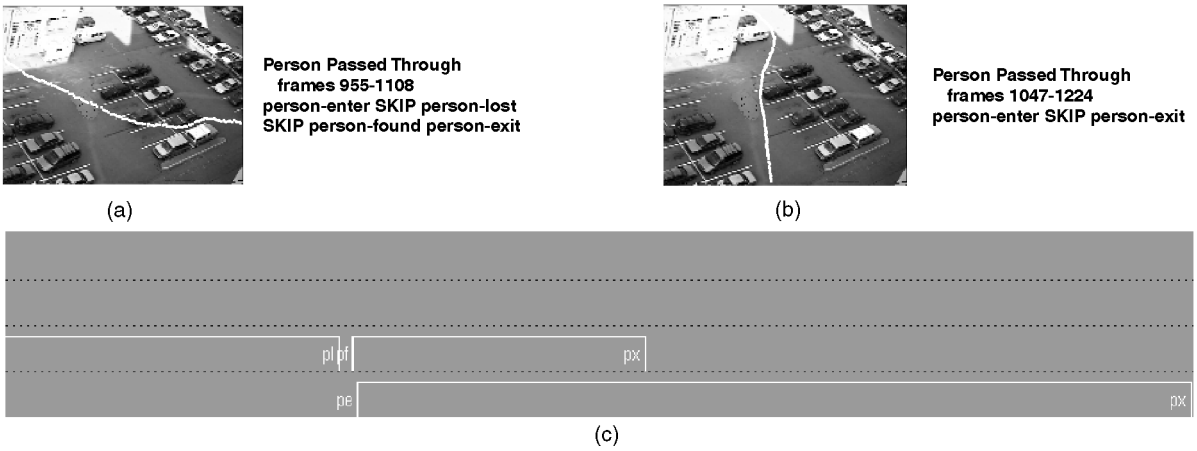


Fig. 16. (a) The track and the labeling of the first person. The broken track is mended by the parser. (b) The track and the labeling of the second person. (c) Temporal extent of the events. `pe`—`person-enter`, `pl`—`person-lost`, `pf`—`person-found`, and `px`—`person-exit`.

	Event	UID	Avg. Size	Class	P	x	y	t	frame	
DROP-OFF	ENTER	724	0.122553	0	0.5	0.450094	0.938069	917907137.8	1906	
	ENTER	665	0.046437	1	0.5	0.6107	0.94674	917907122.5	1799	
	PERSON-LEAVE	665	0.045869	1	0.997846	0.648089	0.98855	917907142.7	1938	
	STOPPED	724		0	0.995784	0.348569	0.345513	917907146.5	1964	
	ENTER	780	0.034293	1	0.5	0.74188	0.980292	917907151.3	1998	
	ENTER	790	0.069093	0	0.5	0.814565	0.032611	917907153.4	2012	
	FOUND	787	0.033573	1	0.5	0.297585	0.357887	917907153.1	2010	
	CAR-LEAVE	790	0.061263	0	0.997285	0.975971	0.211984	917907155.3	2025	
	PERSON-LEAVE	780	0.038616	1	0.999923	0.974494	0.865237	917907158.6	2047	
	PERSON-LEAVE	787	0.032045	1	0.999997	0.296519	0.183704	917907158.7	2048	
	ENTER	813	0.034776	1	0.5	0.012821	0.348379	917907160.9	2063	
	ENTER	816	0.093513	0	0.5	0.960425	0.793899	917907161.9	2070	
	CAR-LEAVE	724	0.097374	0	0.993211	0.972272	0.693728	917907165.2	2091	
	CAR-LEAVE	816	0.089424	0	0.99023	0.693699	0.990798	917907165.2	2091	DRIVE-IN

Fig. 17. Results of track mapping on one of the runs of the system. Two subsets of events, outlined in the picture, correspond to `DRIVE-IN` and `DROP-OFF`. Interpretation of this data is shown in Fig. 18.

where $\mathbf{l}$ is a "low mark" attribute group and $\mathbf{h}$ is the "high mark" group. The attributes contain the frame number, the time-stamp, the position of the object in the image, and its velocity.

As a cost function, we chose a one-sided exponential which "softly" penalizes tracks separated by large spatial distance, but completely rejects fragments overlapping in time. First, we predict position of the object, $\mathbf{r}_p$, based on the $\mathbf{h}$ attribute of the state being completed and the $\mathbf{l}$ attribute of the completing state:

$$\mathbf{r}_p = \mathbf{r}_1 + d\mathbf{r}_1(t_2 - t_1), \tag{19}$$

where $\mathbf{r}_1 = (x_1, y_1)^T$ and $d\mathbf{r}_1 = (dx_1, dy_1)^T$—position and velocity at the end of the first track, t_1 the time at the end of the first track, and t_2, the time at the start of the second.

The penalty function is then computed, based on the distance between the predicted position, $\mathbf{r}_p$, and the position of the object as indicated by the low mark of the candidate completing state, $\mathbf{r}_2$:

$$f(\mathbf{r}_p, \mathbf{r}_2) = \begin{cases} 0, \ if \ (t_2 - t_1) < 0 \\ \exp\left(\frac{(\mathbf{r}_2 - \mathbf{r}_p)^T (\mathbf{r}_2 - \mathbf{r}_p)}{\theta}\right), \ o/w. \end{cases} \tag{20}$$

The scale parameter, θ, was found experimentally.

A partial listing of the grammar employed by our system is shown in Fig. 12. The high-level nonterminals are `CAR-THROUGH`, `PERSON-THROUGH`, `PERSON-IN`, `CAR-OUT`, `CAR-PICK`, and `CAR-DROP`. The production rule probabilities have been manually set to plausible values for this domain. In the conclusion, we discuss the sensitivity of the system to the settings of these probabilities.

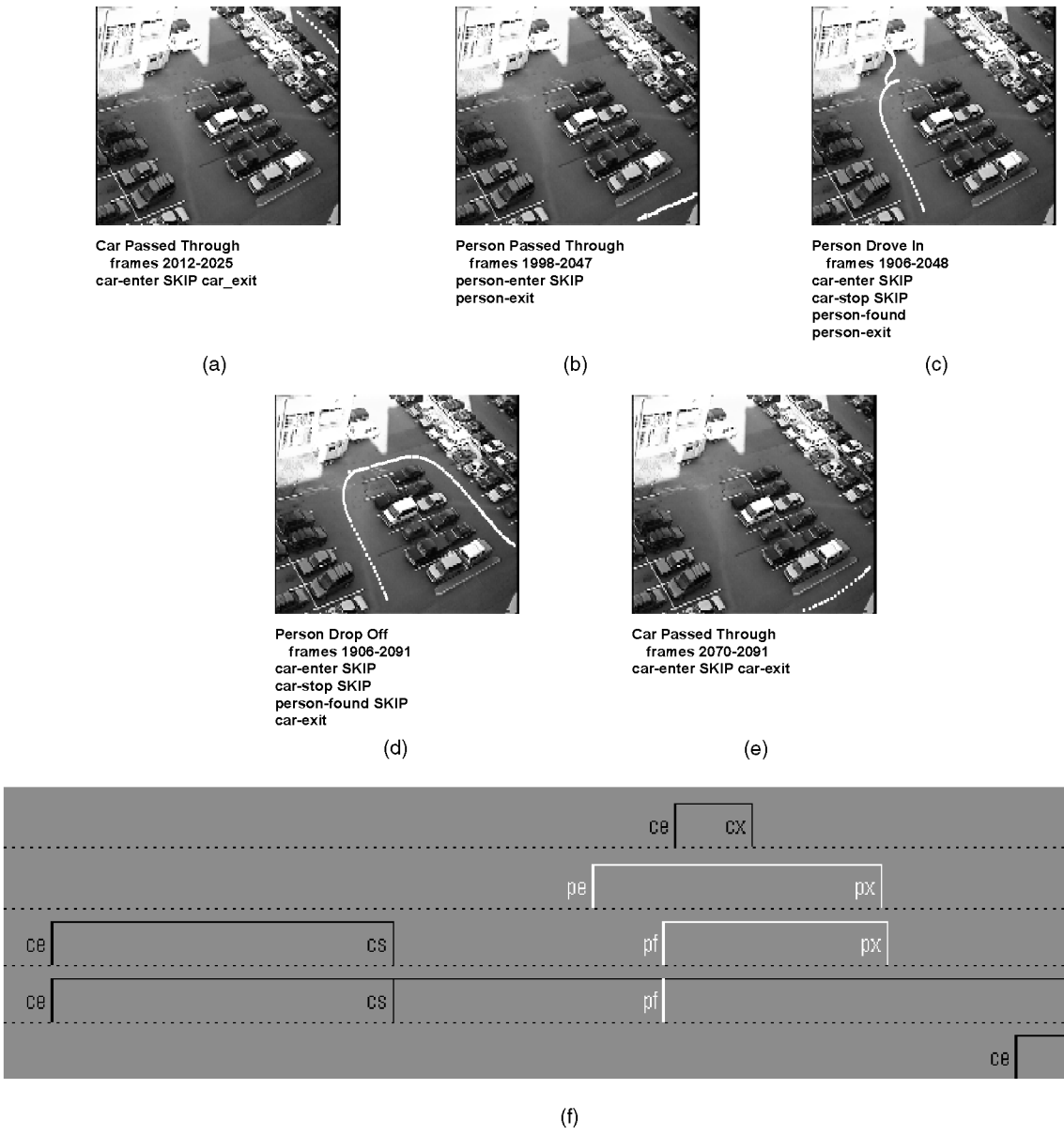


Fig. 18. (a) A car passed through the scene while DROP-OFF was performed. Corresponding track is shown by sequence of white pixels. (b) Person passing through. (c) A person left the car and exited the scene. At this moment, the system has enough information to emit the DRIVE-IN label. (d) The car leaves the scene. The conditions for DROP-OFF are now satisfied and the label is emitted. (e) Before the car performing the DROP-OFF exits the scene, it yields to another car passing through, which is shown here. (f) Temporal extent of actions shown in (a)-(e). Again, actions related to people are shown in white, while vehicle-related primitives are shown in black. Top line of the picture corresponds to the (a), the bottom one to (e). The figure clearly demonstrates concurrency of events. In this figure, primitive events are abbreviated as follows: ce—car-enter, cs—car-stop, cx—car-exit, pe—person-enter, pf—person-found, and px—person-exit.

6.2 Detecting Interactions

We show results of the system run on a data collected on a parking lot at Carnegie Mellon University. The system runs in realtime processing data from a live video feed or a video tape. The tracker and the event generator run on a 175 MHz R10000 SGI O2 machine. The parser runs on a 200 MHz R4400 SGI Indy.

The tracker runs at approximately 12 fps on 160×120 images. It generally exhibited unbroken tracks except in cases of occlusions and rapid lighting changes. The events were mapped using a hand-coded, probabilistic classifier for object type (e.g., car or person), which primarily considers the aspect ratio of the object.

The test data consisted of approximately 15 minutes of video, showing several high-level events such as DROP-OFF and PICKUP. The events were staged in the real environment where the real traffic was present concurrently with the staged events. The only reason for staging the events was to have more examples within 15 minutes of video. The drop-offs and pickups were performed by people unfamiliar with the system. The resulting parses were output in real time.

Fig. 13 shows the event sequence for a PICKUP event, which contains an interaction between a person and a car. Fig. 14a shows the resulting interpretation. Note that, due to lighting changes and video noise, the car was twice lost by

the tracker. The partial tracks are reassembled by the parser to form a consistent interpretation. The bottom row of Fig. 14b displays the temporal extent of the interaction and illustrates that the person and vehicle activities are concurrent, yet are interpreted as parts of one interaction.

A different problem is shown in Figs. 15 and 16. Here, two people crossed the scene. A tracker momentarily lost a person while another one was entering the scene. The concurrent tracks are interpreted as separate noninteracting objects and the break in the complete track is recovered.

The system performs well in a cluttered scene. To demonstrate this, we show a sequence of interpretations which the system produced while observing a DROP-OFF. The sequence shown in Figs. 17 and 18a-18e, demonstrates the capability of the system to parse concurrent activities and interactions in a relatively "busy" environment. While monitoring the scene, the system detected a DROP-OFF as well as several other activities: two instances of CAR-THROUGH and a PERSON-THROUGH event. Fig. 18f shows the temporal extent of activities, shown iconically in Figs. 18a-18e.

All of these parses can be traced down to primitives, which hold the track data. Consequently, the complete track is reconstructed, as shown by white traces in Figs. 18a-18e. In the longest segment of the video, the event generator produces between 150 and 200 events; the exact count depends upon the reaction of the tracker to video noise. After tuning the environment model used by the event generator to convert tracks to events, all high-level interactions were correctly detected.

7 Conclusion

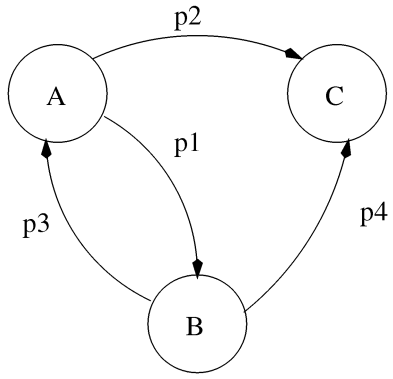
7.1 Summary

This paper describes a probabilistic syntactic approach to the detection and recognition of temporally extended activities and interactions between multiple agents. The fundamental idea is to divide the recognition problem into two levels. The lower level is performed using standard independent probabilistic temporal event detectors, such as hidden Markov models, to propose candidate detections of low-level temporal features. Outputs of these detectors provide the input stream for the stochastic context-free parser. The grammar and parser provide longer range temporal constraints, disambiguate uncertain low-level detections, and allow the inclusion of a priori knowledge about the structure of temporal events in a given domain.

Primary technical contributions of this work are:

1. Extending stochastic context-free parsing to handle uncertainty in the input symbol stream;
2. Providing a general consistency filtering mechanism to enforce intersymbol constraints, such as requiring temporal consistency between primitives; and
3. Extending the consistency filtering to maintain consistent multiobject interactions.

We have developed a real-time system and demonstrated the approach in several experiments on gesture recognition and in visual surveillance. In the surveillance application, we show how the system correctly interprets activities of multiple, interacting objects.



$$
\begin{array}{lcll}
G_1\text{:} & & & \\
A & \rightarrow & BB & [p_1] \\
 & \rightarrow & CB & [p_2] \\
B & \rightarrow & AB & [p_3] \\
 & \rightarrow & C & [p_4] \\
C & \rightarrow & a & [p_5]
\end{array}
\qquad
\left\| \quad
\begin{vmatrix}
0 & p_1 & p_2 \\
p_3 & 0 & p_4 \\
0 & 0 & 0
\end{vmatrix}
\right.
$$

Fig. 19. Left Corner Relation graph of the grammar G_1. Matrix $\mathbf{P}_L$ is shown on the right of the productions of the grammar.

7.2 Comment on Rule Probabilities

This paper does not directly address the problem of computing rule probabilities for SCFGs used in experiments. In all experiments, values of rule probabilities were set to some values that we found plausible. In setting those probabilities, our goal was not estimating them from data, but, rather, expressing expert's beliefs about how typical is the application of each rule. This is reflected, for instance, in weighing rules applied to simple events, such as PERSON-THROUGH, designating a single person walking through the scene, slightly above those related to interactions. In our applications, these values did not have any dramatic effects on the resulting parse and were generally set to $1/n$, where n is the number of productions with the same lefthand side. In most experiments, a uniform distribution over the rules was sufficient. However, setting values of rule probabilities to represent a uniform distribution is not a simple task in the general case, primarily due to the exact form of the grammar and recursion.

If enough labeled data is available, these probabilities can be estimated by a number of known methods, based on a Baum-Welch estimation procedure, such as the *inside-outside* algorithm. The estimation of rule probabilities is beyond the scope of this paper. For a reference, we suggest the reader to refer to [24], [19], or [11]. For the review of techniques for grammatical inference, Lee [22] provides a good reference and bibliography.

The only particularly sensitive probability setting in our system is the $SKIP$ production. As noted, that parameter needs to be tuned to the application depending upon the false alarm rate of the low-level detectors. While it should be possible to learn that parameter automatically from training data, we have not systematically explored that problem.

G_2:

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

$\mathbf{P}_L$	S	A	B	C
S		p_1		p_2
A		p_4		
B				
C			$p_8 + p_9$	

$\mathbf{R}_L$	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

Fig. 20. Left Corner ($\mathbf{P}_L$) and its Reflexive Transitive Closure ($\mathbf{R}_L$) matrices for a simple SCFG.

7.3 Behavior as Context

Finally, we comment on a broader characterization of the work presented here, namely *behavior as context.* Consider the surveillance application and suppose a blob is detected, but, because of imaging noise or occlusion, it cannot be identified well as a car or a person. Further, assume the blob stops in the parking area, and a well-identified person moves away from the first blob, and the first blob just remains in the parking area. The system has just observed a `DRIVE-IN` and, because of the behavior, can now more reliably label the blob a car. By providing an activity recognition system that explicitly models behavior and separates it from detection, we can view the behavior as the disambiguating context applied to the result of detection.

APPENDIX A
PREDICTION STEP: COMPUTING R_L

During prediction, the parsing algorithm descends the grammar down to the leaf nodes hypothesizing the next possible production to use. This may result in predicting a state more than once. 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 nonterminal A is considered for expansion and given productions for A

$$
\begin{array}{l}
A \rightarrow Aa \\
\quad \rightarrow a
\end{array}
$$

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 nonprobabilistic Earley parser, it normally means that no further expansion of the production should be made and no duplicate states should be added. In the 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 [20].

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 nonterminals can be computed in a matrix form. The form of the recursive correction matrix $\mathbf{R}_L$ can be derived using a simple example presented in Fig. 19. The graph of the relation presents direct left corner relations between nonterminals of G_1. The LC Relation matrix $\mathbf{P}_L$ 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

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

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,

$$
\begin{aligned}
P(a) = P(Y \rightarrow a\eta) \quad & \sum_{\forall X} P(X \overset{*}{\Rightarrow} Y) \\
= P(Y \rightarrow a\eta) \quad & (P(X \overset{0}{\Rightarrow} Y) + \\
& P(X \overset{1}{\Rightarrow} Y) + \\
& P(X \overset{2}{\Rightarrow} Y) + \ldots,
\end{aligned}
\tag{21}
$$

where $P(X \overset{k}{\Rightarrow} Y)$ 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, $\mathbf{P}_L$ (Fig. 19), as kth power of it, $\mathbf{P}_L^k$. And, the complete reflexive transitive closure $\mathbf{R}_L$ is a matrix of infinite sums that has a closed form solution:

G_2:

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

$\mathbf{P}_U$	S	A	B	C
S				p_2
A				
B				
C			p_9	

$\mathbf{R}_U$	S	A	B	C
S	1		p_2p_9	p_2
A		1		
B			1	
C			p_9	1

Fig. 21. Unit Production ($\mathbf{P}_U$) and its Reflexive Transitive Closure ($\mathbf{R}_U$) matrices for a simple SCFG.

$$\mathbf{R}_L = \mathbf{P}_L^0 + \mathbf{P}_L^1 + \mathbf{P}_L^2 + \ldots = \sum_{k=0}^{\infty} \mathbf{P}_L^k = (\mathbf{I} - \mathbf{P}_L)^{-1}. \quad (22)$$

Thus, the correction to the completion step should be applied, by first, descending the chain of left corner productions, indicated by a nonzero value in $\mathbf{R}_L$:

$$\begin{cases} i : X_k \rightarrow \lambda.Z\mu \quad [\alpha, \gamma] \\ \forall Z \quad s.t. \quad \mathbf{R}_L(Z,Y) \neq 0 \Rightarrow i : Y \rightarrow .\nu \quad [\alpha', \lambda'] \\ Y \rightarrow \nu \end{cases} \quad (23)$$

and then correcting the forward and inner probabilities for the left recursive productions by an appropriate entry of the matrix $\mathbf{R}_L$:

$$\begin{aligned} \alpha' &= \sum_{\lambda,\mu} \alpha(i : X_k \rightarrow \lambda.Z\mu)\mathbf{R}_L(Z,Y)P(Y \rightarrow \nu) \\ \gamma' &= P(Y \rightarrow \nu). \end{aligned} \quad (24)$$

Matrix $\mathbf{R}_L$ can be computed once for the grammar and used at each iteration of the prediction step (Fig. 20).

Appendix B

Completion Step: Computing R_U

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

$$\begin{array}{lcl} A & \rightarrow & \mathrm{B} \\ & \rightarrow & \mathrm{a} \\ B & \rightarrow & \mathrm{A} \end{array}$$

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

$$\begin{array}{lcl} j : A_j & \rightarrow & .B \\ j : A_j & \rightarrow & .a \\ j : B_j & \rightarrow & .A. \end{array}$$

Among others, this operation will produce another state $i : A_j \rightarrow a.$, which will cause the parse to go into infinite loop. In nonprobabilistic Earley parser, that would mean that we just simply do not add the newly generated states and proceed with the rest of them. However, here 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* [35].

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 $\mathbf{R}_U$ (Fig. 21), considering the Unit Production relation, expressed by a matrix $\mathbf{P}_U$. $\mathbf{R}_U$ is found as $\mathbf{R}_U = (\mathbf{I} - \mathbf{P}_U)^{-1}$. The resulting expanded completion algorithm accommodates the recursive loops:

$$\begin{cases} j : X_k \rightarrow \lambda.Z\mu \quad [\alpha, \gamma] \\ \forall Z \quad s.t. \quad \mathbf{R}_U(Z,Y) \neq 0 \Rightarrow i : X_k \rightarrow \lambda Z.\mu \quad \alpha', \gamma'] \\ i : Y_j \rightarrow \nu. \quad [\alpha'', \gamma''], \end{cases} \quad (25)$$

where computation of α' and γ' is corrected by a corresponding entry of $\mathbf{R}_U$:

$$\begin{aligned} \alpha' &= \sum_{\nu} \alpha(i : X_k \rightarrow \lambda.Z\mu)\mathbf{R}_U(Z,Y)\gamma''(i : Y_j \rightarrow \nu.) \\ \gamma' &= \sum_{\nu} \gamma(i : X_k \rightarrow \lambda.Y\mu)\mathbf{R}_U(Z,Y)\gamma''(i : Y_j \rightarrow \nu.). \end{aligned} \quad (26)$$

As $\mathbf{R}_L$, unit production recursive correction matrix $\mathbf{R}_U$ can be computed once for each grammar.

Acknowledgments

The authors thank Chris Stauffer and Eric Grimson of the MIT AI lab for graciously providing the tracker and modifying it as necessary to generate events and to handle the data of the CMU experiment. This work was supported in part by DARPA contract DAAL01-97-K-0103.

References

[1] J.K. Aggarwal and Q. Cai, "Human Motion Analysis: A Review," *Computer Vision and Image Understanding*, vol. 73, no. 3, pp. 428–440, 1999.

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

[3] A.V. Aho and T.G. Peterson, "A Minimum Distance Error-Correcting Parser for Context-Free Languages," *SIAM J. Computing*, vol. 1, pp. 305–312, 1972.

[4] M.L. Baird and M.D. Kelly, "A Paradigm for Semantic Picture Recognition," *Pattern Recognition*, vol. 6, no. 1, pp. 61–74, 1974.

[5] A.F. Bobick, "Movement, Activity, and Action: The Role of Knowledge in the Perception of Motion," *Philosophical Trans. Royal Soc. London B*, pp. 1,257–1,265, 1997.

[6] A.F. Bobick and A.D. Wilson, "A State-Based Approach to the Representation and Recognition of Gesture," *IEEE Trans. Pattern Analysis and Machince Intelligence*, vol. 19, no. 12, pp. 1,325–1,337, Dec. 1997.

[7] T.L. Booth and R.A. Thompson, "Applying Probability Measures to Abstract Languages," *IEEE Trans. Computers*, vol. 22, no. 5, pp. 442–450, May 1973.

[8] M. Brand, "Understanding Manipulation in Video," *Proc. Second Int'l Conf. Automatic Face and Gesture Recognition '96*, pp. 94–99, 1996.

[9] M. Brand and N. Oliver, "Coupled Hidden Markov Models for Complex Action Recognition," *Computer Vision and Pattern Recognition*, pp. 994–999, 1996.

[10] F. Bremond and G. Medioni, "Scenario Recognition in Airborne Video Imagery," *Proc. Workshop Interpretation of Visual Motion*, pp. 57–64, 1998.

[11] T. Briscoe and N. Waegner, "Robust Stochastic Parsing Using Inside-Outside Algorithm," *Proc. Am. Assoc. Artificial Intelligence Workshop Statistically-Based Natural Language Programming Techniques*, 1992.

[12] H. Bunke and D. Pasche, "Parsing Multivalued Strings and its Application to Image and Waveform Recognition," *Structural Pattern Analysis,* 1990.
[13] E. Charniak, *Statistical Language Learning.* Cambridge, Mass. and London: MIT Press, 1993.
[14] J.D. Courtney, "Automatic Video Indexing via Object Motion Analysis," *Pattern Recognition,* vol. 30, no. 4, pp. 607–625, 1997.
[15] E. Csunhaj-Varju, J. Dassow, J. Kelemen, and G. Paun, *Grammar Systems: A Grammatical Approach to Distribution and Cooperation.* Gordon and Breach Science Publishers, 1994.
[16] T.J. Darrell and A.P. Pentland, "Space-Time Gestures," *Proc. Conf. Computer Vision and Pattern Recognition,* pp. 335–340, 1993.
[17] J.C. Earley, "An Efficient Context-Free Parsing Algorithm," PhD thesis, Carnegie-Mellon Univ., 1968.
[18] W.E.L. Grimson, C. Stauffer, R. Romano, and L. Lee, "Using Adaptive Tracking to Classify and Monitor Activities," *Computer Vision and Pattern Recognition,* pp. 22–29, 1998.
[19] F. Jelinek, J.D. Lafferty, and R.L. Mercer, "Basic Methods of Probabilistic Context Free Grammars," *Speech Recognition and Understanding. Recent Advances, Trends, and Applications,* Pietro Laface Mori and Renato Di, eds., vol. F75 of *NATO Advanced Study Institute,* pp. 345–360, Berlin: Springer Verlag, 1992.
[20] F. Jelinek and J.D. Lafferty, "Computation of the Probability of Initial Substring Generation by Stochastic Context Free Grammars," *Computational Linguistics,* vol. 17, no. 3, pp. 315–323, 1991.
[21] F. Jelinek, *Statistical Methods for Speech Recognition.* Cambridge, Mass.: MIT Press, 1999.
[22] L Lee, "Learning of Context-Free Languages: A Survey of the Literature," Technical Report TR-12-96, Harvard Univ., 1996.
[23] R. Narasimhan, "Labeling Schemata and Syntactic Descriptions of Pictures," *InfoControl,* vol. 7, no. 2, pp. 151–179, 1964.
[24] H. Ney, "Stochastic Grammars and Pattern Recognition," *Speech Recognition and Understanding,* P. Laface and R. DeMori, eds., pp. 319–344, 1992.
[25] N. Oliver, B. Rosario, and A. Pentland, "Statistical Modeling of Human Interactions," *Proc. Computer Vision and Pattern Recognition, The Interpretation of Visual Motion Workshop,* pp. 39–46, 1998.
[26] B.J. Oomen and R.L. Kashyap, "Optimal and Information Theoretic Syntactic Pattern Recognition for Traditional Errors," *Lecture Notes in Computer Science,* vol. 1,121, pp. 11–20, Springer, 1996.
[27] L.R. Rabiner and B.H. Juang, *Fundamentals of Speech Recognition.* Englewood Cliffs: Prentice Hall, 1993.
[28] M. Rudolf, *The Grammar of Conducting: A Comprehensive Guide to Baton Techniques and Interpretation.* New York: Schimmer Books, 1994.
[29] A. Sanfeliu and R. Alquezar, "Efficient Recognition if a Class of Context-Sensitive Languages Described by Augmented Regular Expressions," *Lecture Notes in Computer Science,* vol. 1,121, pp. 1–10, Springer, 1996.
[30] A. Sanfeliu and M. Sainz, "Automatic Recognition of Bidimensional Models Learned by Grammatical Inferrence in Outdoor Scenes," *Lecture Notes on Computer Science,* vol. 1,121, pp. 160–169, Springer, 1996.
[31] R. Schalkoff, *Pattern Recognition: Statistical, Structural, and Neural Approaches.* New York: Wiley, 1992.
[32] J. Schlenzig, E. Hunter, and R. Jain, "Recursive Identification of Gesture Inputs Using Hidden Markov Models," *Proc. Second Ann. Conf. Applications of Computer Vision,* pp. 187–194, 1994.
[33] T. Starner, J. Weaver, and A. Pentland, "Real-Time American Sign Language Recognition Using Desk and Wearable Computer-Based Video," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 20, no. 12, Dec. 1998.
[34] T.E. Starner and A. Pentland, "Visual Recognition of American Sign Language Using Hidden Markov Models," *Proc. Int'l Workshop Automatic Face- and Gesture-Recognition,* pp. 189–194, 1995.
[35] A. Stolcke, "An Efficient Probabilistic Context-Free Parsing Algorithm That Computes Prefix Probabilities," *Computational Linguistics,* vol. 21, no. 2, pp. 165–201, 1995.
[36] M.G. Thomason, "Stochastic Syntax-Directed Translation Schemata for Correction of Errors in Context-Free Languages," *IEEE Trans. Computers,* vol. 24, pp. 1,211–1,216, 1975.
[37] W.G. Tsai and K.S. Fu, "Attributed Grammars—A Tool for Combining Syntatctic and Statistical Approaches to Pattern Recognition," *IEEE Trans. Systems, Man, and Cybernetics,* vol. 10, no. 12, pp. 873–885, 1980.
[38] A.D. Wilson, A.F. Bobick, and J. Cassell, "Temporal Classification of Natural Gesture and Application to Video Coding," *Proc. Conf. Computer Vision and Pattern Recognition,* pp. 948–954, 1997.
[39] C. Wren, A. Azarbayejani, T. Darrell, and A. Pentland, "Pfinder: Real-time Tracking of the Human Body," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 19, no. 7, pp. 780–785, July 1997.
[40] J. Yamato, J. Ohya, and K. Ishii, "Recognizing Human Action in Time-Sequential Images Using Hidden Markov Model," *Proc. Conf. Computer Vision and Pattern Recognition,* pp. 379–385, 1992.
[41] S.J. Young, N.H. Russell, and J.H.S. Thornton, "Token Passing: A Simple Conceptual Model for Connected Speech Recognition Systems," Technical Report CUED/F-INFENG/TR. 38, Univ. of Cambridge, July 1989.

Yuri A. Ivanov is pursuing a PhD degree in media arts and sciences at the Massachussetts Institute of Technology Media Laboratory. He received the MS degree in media arts and sciences from MIT in 1998 and the MS degree in electrical engineering and computer science from the St. Petersburg Academy of Air and Space, Russia, in 1992. His main research intrerest is machine vision with emphasis on action understanding, machine learning, and perceptual interfaces.

Aaron F. Bobick received the PhD degree in cognitive science from the Massachusetts Institute of Technology in 1987 and also holds BS degrees from MIT in mathematics and computer science. In 1987, he joined the Perception Group of the Artificial Intelligence Laboratory at SRI International and soon after was jointly named a visiting scholar at Stanford University. From 1992 until July 1999, he served as an assistant and then associate professor in the Vision and Modeling Group of the MIT Media Laboratory. He has recently moved to the College of Computing at the Georgia Institute of Technology, where he is an associate professor in the GVU and Future Computing Environments laboratories.

Dr. Bobick has performed research in many areas of computer vision. His primary work has focused on video sequences where the imagery varies over time either because of change in camera viewpoint or change in the scene itself. He has published papers addressing many levels of the problem from validating low-level optic flow algorithms to constructing multirepresentational systems for an autonomous vehicle to the representation and recognition of high-level human activities. The current emphasis of his work is on action understanding where the imagery is of a dynamic scene and the goal is to describe the action or behavior. Three examples are the basic recogniton of human movments, natural gesture understanding, and the classification of football plays. Each of these examples require describing human activity in a manner appropriate for the domain and developing recognition techniques suitable for those representations.