Proc. 28th Conference on Computer Communications, INFOCOM 09, Rio de Janeiro, Brazil, March 2009, pp. 900-908.

Event Recognition in Sensor Networks by Means of Grammatical Inference

Sahin Cem Geyik and Boleslaw K. Szymanski
Department of Computer Science and Center for Pervasive Computing and Networking
Rensselaer Polytechnic Institute
Troy, New York 12180
{geyiks,szymansk}@cs.rpi.edu

***Abstract*—Modern military and civilian surveillance applications should provide end users with the high level representation of events observed by sensors rather than with the raw data measurements. Hence, there is a need for a system that can infer higher level meaning from collected sensor data. We demonstrate that probabilistic context free grammars (PCFGs) can be used as a basis for such a system. To recognize events from raw sensor network measurements, we use a PCFG inference method based on Stolcke(1994) and Chen(1996). We present a fast algorithm for deriving a concise probabilistic context free grammar from the given observational data. The algorithm uses an evaluation metric based on Bayesian formula for maximizing grammar a posteriori probability given the training data. We also present a real-world scenario of monitoring a parking lot and the simulation based on this scenario. We described the use of PCFGs to recognize events in the results of such a simulation. We finally demonstrate the deployment details of such an event recognition system.**

I. Introduction

Wireless sensor networks(WSN) are often deployed to collect and process useful information about their surroundings. They are composed of sensor nodes that measure the environment, base station(s) that collect information sent by the sensor nodes and relay nodes which transmit the data efficiently from sensor nodes to the base station. A detailed survey of wireless sensor networks can be found in [1].

In today's sensor applications that collect vast amounts of measurement data, usually too big for manual inspection, it becomes more and more important for users to receive a high level representation of raw measurements. Systems that are capable of summarizing the information into a compact and meaningful form (compared to raw signal data) are needed.

Acquiring higher level meaning from sensor network measurements is called root cause analysis and several machine learning techniques have been used to achieve it. The focus of this paper is on using Probabilistic Context Free Grammars (PCFG) for event recognition. While it is certainly possible to construct grammars manually, such a process often suffers from problems of finding the right level of abstraction, with the dangers of either over-generalization (a grammar accepting nearly every string) or under-generalization (a grammar accepting only the training data). To address this problem, we are presenting here a grammar induction method inspired by the scheme introduced by Stolcke [2]. There are two differences between his approach and ours. First, each method uses the different metric for evaluating the grammar at each step of its construction. Second, we are using a novel method for fast calculation of the metric that takes advantage of the properties of the chunk and merge operations, originally introduced by Stolcke. Please note that what we are presenting in this paper is actually a $supervised$ learning method because the definitions of the events to be recognized are initially given to the learning algorithm. Then, the recognition phase takes place during which new data are classified into events using the grammar created in the learning stage.

The rest of the paper is organized as follows. We start with a basic introduction to PCFGs. The next section describes the previous work on PCFG inference methods and applications of PCFGs to event detection in WSNs. Then, we define our method for grammar inference and analyze its complexity. Finally, we present a real world scenario of parking lot monitoring and its simulations, as well as deployment details of such a system on a sensor network.

II. Probabilistic Context Free Grammars

A Probabilistic Context Free Grammar consists of a five-tuple $<S_{nt},S_t,R,Pr,Start>$ where:

- S_{nt} is a list of nonterminal symbols,
- S_t is a list of terminal symbols,
- R is a list of production rules that define how terminal and nonterminal symbols can be generated from nonterminal symbols or, equivalently, how a string of terminal and nonterminal symbols can be reduced to a nonterminal symbol,
- Pr is a list of probabilities, each assigned to a rule to define the production probability of that rule as opposed to the other rules defining the same nonterminal.

Fig. 1 shows a simple probabilistic context free grammar which outputs strings of type $\mathbf{a^n\ b^n}$ and of the average length of 10 terminal symbols.

START ⟶ B (1.0)
B ⟶ a B b (0.8) | a b (0.2)

Fig. 1. A Very Simple PCFG

The probability of a sentence is defined by the product of probabilities of productions applied at the branches of the

parsing tree of that sentence. In Fig. 1, the string *a a a b b b* has the probability of $1.0 * 0.8 * 0.8 * 0.2 = 0.128$.

Probabilistic Context Free Grammars have many uses in speech recognition [15], [16], natural language processing [18], computational biology [17], sensor networks [4], [8], [10], [11] etc.

III. Previous Work

In this section, we present a brief overview of literature on PCFG inference and application of PCFGs to sensor data.

A. Event Recognition using PCFGs

There are several papers discussing PCFG application to sensor network data. In [4], the authors use a Probabilistic Context Free Grammar to parse the actions of a user to infer higher level behaviors. Clearly, the same behavior can be achieved in different ways, so the approach proposed in that work assigns a probability to different combinations of events constituting certain behavior. The paper also presents a way to calculate the probability of an event for a given set of actions.

Papers [5] and [6] complement each other. The first one introduces Address-Event Imagers for sensor nodes. These are image sensors, capabilities of which are limited to achieve efficiency in power, computation cost, storage, communication bandwidth and also to promote privacy. The paper describes the working policies, available platforms and programming details for this kind of a system. The paper concludes with several examples of pattern recognition in data collected by Address-Event Imagers. The second paper describes how an assisted living application can be implemented using the Address-Event Imagers. It describes how cooking differs from cleaning the dishes in the kitchen in terms of positions of the agent and how this difference can be captured by PCFGs.

Following the methodology of [6], the authors describe in [7] a human behavior parsing sensor system from start to end. The paper discusses a context free grammar that distinguishes cooking from cleaning and from other kitchen activities. Experimental results are presented together with the grammars themselves which are defined in a varying hierarchy of event recognition steps.

In [8], the authors are using a visual system to recognize human gestures and to detect interactions in a parking lot environment. Authors utilize an Earley-Stolcke parser [2], [9] to parse sequences into actions. A run-time incremental parsing is implemented in which the states (in Earley Model) whose probabilities drop below a certain value are removed. Also, only sequences of events up to a fixed size are considered for parsing (in effect limiting parsing to a sliding window).

In [10], PCFGs are used to recognize complex, multitasked activities from videos of rounds of game of Blackjack. The paper introduces new parsing strategies to enable error detection and recovery assuming that a sequence irreducible during Earley-Stolcke parsing is caused by an error of one of three types: (i) insertion (in which case the currently parsed token is ignored), (ii) substitution (in such a case, parsing continues on the substituted token), or (iii) deletion (in that case, parsing continues on insertion of the deleted token).

In [11], the authors transform hierarchical PCFG into a hierarchical Bayesian Network and use deleted interpolation (DI) [18] to combine two recognition models, one of which is precise but unreliable while the other is less precise but more reliable. An advantage of such an approach is its ability to detect overlapping activities.

B. PCFG Inference

There are two different tasks associated with the PCFG inference problem. The first one arises when we know in advance the rules of the underlying grammar. In that case, the problem is to estimate the probabilities for the rules from the training data. The so-called inside-outside algorithm serves as the basis for the general solution of this task [12], [13].

Constructing PCFGs from scratch, however, requires alternative learning techniques. There are two different approaches to this problem. The first one uses two operators: merge and chunk [2], [14]. This is the approach that forms the basis for our inference algorithm. This scheme utilizes Bayesian formulation to evaluate the grammar. The solution is obtained by maximizing a priori values for the model topology and parameter settings. Our approach uses a different and simpler evaluation function and a novel, fast method for computing the Bayesian metric taking advantage of properties of merge and chunk operations.

The second approach, described in [3], starts with a grammar which creates all possible strings. Then, the grammar is made specific to the training data by using five different moves: concatenation, classing, repetition, smoothing and specialization. This approach, like the previous one and the one that we used in our implementation, also utilizes a Bayesian evaluation of the grammar.

IV. Grammar Inference

In this section, we explain how our grammar inference scheme works. We start by introducing the operations that, step by step, construct a PCFG. Then, we define the Bayesian metric for PCFG evaluation and describe how to compute it efficiently by taking advantage of the properties of the operators that are used in the grammar inference.

A. Operations Used for Grammar Construction

This section explains the method introduced initially by Stolcke [2]. Grammar construction consists of two steps: sample incorporation, and application of operators.

1) Sample Incorporation: Sample incorporation is the initial step of constructing the grammar from training data. The training data is a set of sentences representing the event that we want to recognize by the grammar. Each sentence is a string of terminal symbols. These terminal symbols are reduced by the grammar to nonterminals of the form:

$$N_i \rightarrow symbol_i \ (count\ of\ this\ symbol)$$

where $count$ of a rule is the number of times the terminal symbol appears in the training data. This is done in order to

separate $terminal$ productions from $nonterminal$ productions. The sentences are reduced in the initial grammar by rules of the form:

$$START \rightarrow N_{s,1} \ ... \ N_{s,j} \ (count \ of \ this \ sentence).$$

It should also be noted that the production count is held, rather than the probabilities, but this is done merely for computational simplicity. Probabilities are easily computed by using the following equation:

$$P(rule_i) = \frac{count(rule_i)}{\sum_{k=1}^{n} count(rule_k)} \quad (1)$$

where n is the number of rules in the nonterminal's definition.

2) Operators: We are using two operators: merge and chunk [2] for building up the grammar step by step. We explain how these two operators work by giving examples.

Merge takes two nonterminals and reduces them into a new nonterminal. The Right Hand Side (RHS) occurrences of these two nonterminals are replaced by the new nonterminal which inherits the rules of these two nonterminals, as well as their counts. The original two merged nonterminals are removed from the grammar. See Fig. 2 for an example.

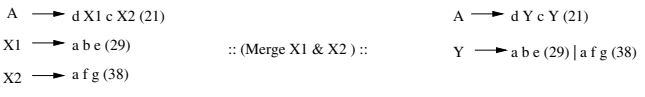

Fig. 2. An Example of Merge Operation

Merge is beneficial because it can create recursion in examples like **X1** → **a X2 b**, when we merge **X1** and **X2**. By definition, an initial grammar cannot contain recursion; after all, the collected evidence is always finite. Hence, such merge is necessary to capture the recursive nature of some components of the event being abstracted by the context free grammar.

There are two special cases that need to be considered during merge execution. In the first case, two different rules for the same nonterminal may become the same because of nonterminal replacements. When such a case occurs, these two rules are combined into one with their counts added. The second special case arises when **START** nonterminal is merged with another nonterminal. The new nonterminal is named START and if the rule **START** → **X2** is present and we are merging START with X2, a rule **START** → **START** would be created. If this happens, such rule is simply deleted, ignoring its count as well, of course.

Another operator, chunk, creates a new nonterminal with a single rule: a string composed of nonterminals in the current grammar. It simply replaces all the occurrences of this string with the new nonterminal. Each time a replacement takes place, the count of the rule is added to the count of the new nonterminal rule. An example of chunk is given in Fig. 3.

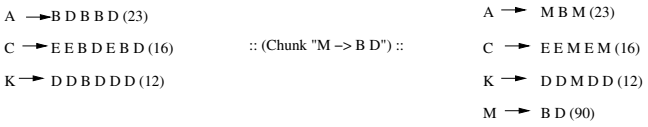

Fig. 3. An Example of Chunk Operation

B. Evaluation Metric for the PCFG

Our goal is to find a grammar G that maximizes a posteriori probability given the training data $O = \{o_1, o_2, \dots\}$. Hence, we want to compute the grammar defined as [2], [3]:

$$G = argmax_G P(G|O). \quad (2)$$

Using Bayes' formula, we obtain the following expansion of Eq. (2).

$$G = argmax_G \frac{P(G)P(O|G)}{P(O)} = argmax_G P(G)P(O|G)$$

where $P(G)$ is the grammar a priori probability which is related to the grammar description length. The shorter is the grammar, the more probable it is amongst other grammars (based on *Occam's Razor* principle). From information theory, we have the following equation for $P(G)$:

$$P(G) = 2^{-l(G)}. \quad (3)$$

$P(O|G)$ stands for the a priori probability of the training data, given the grammar G. It is calculated by multiplying the probabilities of all sentences in the grammar:

$$P(O|G) = \prod_{i=1}^{|O|} p(o_i|G). \quad (4)$$

To define the description length of the grammar, $l(G)$, we need a method for the canonical representation of a grammar. For simplicity, we are using a representation which is easy to understand and easy to implement. For each nonterminal in the system, $l(G)$ increases by one (corresponding to the nonterminal name). For each rule in the nonterminal, $l(G)$ increases by $1 + (number \ of \ nonterminals \ in \ the \ rule)$, where 1 accounts for the separation symbol. We have used a fixed equal bit length notation for each symbol in the grammar which gave us the above formulation for the description length. Better and shorter representations can be found, however, our problem is not actually finding a better representation, but finding a better grammar, given a representation.

C. Computation of the Evaluation Metric

Chunk and merge operations are applied in hopes of increasing the a posteriori value defined by Eq. (2). Therefore, an efficient algorithm for calculating a posteriori value after each operation should be devised. Below, we describe the heuristics that we are using for such calculations.

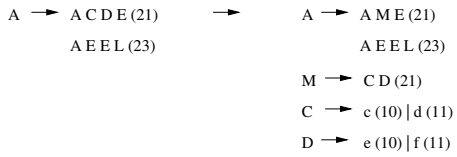

Fig. 4. Another Chunk Example

1) Chunk: Calculating $P(G)$ is easy, given the modified grammar. However, it is also easy to see that $P(O|G)$ does not change as a result of applying a chunk operation.

In the example given in Fig. 4, any reduction to nonterminal A applied in the parsing tree of any sentence in the training data is based in either *A C D E* with probability 21/44 or in *A E E L* with probability 23/44. Chunking *C D* does not change this situation. The only thing that changes is the number of nodes in the parsing tree, but the probabilities stay the same. A simple generalization of this argument would demonstrate that chunk does not change probabilities of the grammar productions.

2) Merge: Calculating $P(G)$ is again easy, given the modified grammar. However, now $P(O|G)$ changes whenever merge operation takes place during modification of the grammar. A naive and inefficient approach would be to re-parse the training data to compute $P(O|G)$. However, we devised a fast heuristic for calculating the new $P(O|G)$ which we illustrate with a simple example. The heuristic utilizes the fact that when the rules of two nonterminals combine in merge, the reductions using those rules have different probabilities within the new setting.

In Fig. 5, the powers represent the number of times the production is used throughout all the training data. In each division of two fractions, the numerator represents the previous value and the denominator the new value for the production. Thanks to the way we calculate $P(O|G)$ in Eq. (4), it is sufficient to compute just the power of the result of the division which means applying the probability changes (due to the new rule probabilities caused by the $merge$ operation) on all parse trees to $P(O|G)$. This gives us a very fast and accurate way of calculating $P(O|G)$ without re-parsing the training data.

D. Search for the Best Merge and Chunk Arguments

Our implementation evaluates any two combinations of nonterminals to find the merge operation with the highest a posteriori value advancement. This is done so because we have not found any faster way to determine which pair of nonterminals will be most beneficial to merge, except the simple heuristic described below.

Search for the best chunk argument grows quickly very large with the growth of grammar size, if we need to check strings of increasing length as arguments. Fortunately, the length of the strings that need to be considered is up to *5*. We show that if there is a string that shrinks the grammar when it is chunked, then it has a substring (possibly itself) with the same property which is no longer than 5 symbols. Indeed, chunking a string which occurs just once cannot shrink the grammar. Each time a replacement (over the grammar representation and not the training data) of a chunk with *k* nonterminals occurs, grammar gets smaller by *k-1* symbols (replacing k symbols with a single symbol which is the name of the new chunk nonterminal). At the same time, introducing a chunk of size k creates a new nonterminal with one rule which increases the size of the grammar by *k+2*. If we denote the number of replacements through grammar representation as *n*, then for chunk to decrease the grammar size, we must have:

$$n(k-1) > k+2 \text{ and for } n = 2 \ ; \ k > 4.$$

If there is a longer chunk that occurs at least twice, then it will shorten the grammar even more than chunking its substring of the length 5, but looking for the repeating strings with lengths up to 5 guarantees that we find a chunk decreasing the grammar length, if there is one. Limiting the search to repeating strings with length up to 5 significantly cuts its complexity.

Our implementation looks through repeating strings of length up to 5 and chooses the one which decreases the size of the grammar the most. Size difference between two grammars is expressed by the following formula, in which *n* is the number of replacements and *k* is the length of the chunk:

$$l(G)_{previous} - l(G)_{new} = (n-1)(k-1) - 3. \qquad (5)$$

E. Inference Algorithm

In most cases, merge operation decreases a posteriori probability according to definition of $P(G|O)$ given by Eq. (2) and its expansion. When, during merge, we replace a production with small number of alternatives by the one with larger number of alternatives (merge in Fig. 5 shows an example of such replacement), the advantage of the shorter grammar length usually does not make up for the decrease in $P(O|G)$. In contrast, a chunk operation always decreases the grammar length and increases its a posteriori probability. Therefore, we expect that advantageous merge operations will be less likely to be encountered than advantageous chunk operations. Consequently, it is beneficial for a posteriori probability of the grammar to execute as many chunk operations as possible before applying any merge operations. Hence, the general form of each of our interference step is

$$chunk^* \ merge$$

where we do all the beneficial chunk operations before doing any merges. An outline for the algorithm is given in Table Algorithm 1.

We will next prove that applying a chunk operation before a merge does not make delayed merge inapplicable for inference.

Theorem 1. *A chunk operation does not eliminate feasible merge operations nor does it change their impact on a posteriori probability of the grammar.*

Proof: A chunk operation increases the number of nonterminals by introducing a new nonterminal which consists

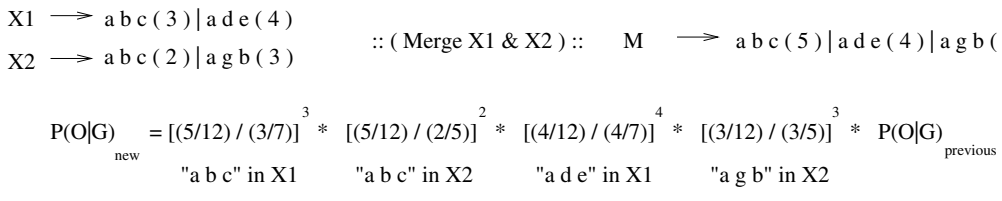

Fig. 5. Calculation of $P(O|G)$ for Merge Operation

Algorithm 1 PCFG Inference Algorithm

```
posterior = 1.0
while true do
  if best chunk shrinks the grammar then
    do chunk , posterior * = gain_from_chunk
  else
    if (posterior * = gain_from_best_merge) > 1.0 then
      do merge, posterior * = gain_from_merge
    else
      output grammar and quit
    end if
  end if
end while
```

of a single rule: a string of nonterminals with probability 1. Accordingly, it increases possibilities for merge (as there are more nonterminals defined) and any merge possible before the chunk took place is still possible afterwards. Proving that a merge possible before the chunk is applied changes a posteriori probability of the grammar the same way, regardless if it is executed before or after the chunk, requires a careful look into how the grammar is changed by the chunk operation.

When a chunk operation occurs, the number of times a nonterminal (and its appropriate rule) is used does not change. Rather, in any parsing tree (which contains the string of nonterminals that is now a *chunk*) of any sentence in the training data, a new node is created with the name of the new nonterminal. From this new nonterminal, however, the parsing continues as it did before the chunking was applied.

A merge operation combines two nonterminals (and their rules) into a new nonterminal. This operation effects $P(O|G)$ because production probabilities change with this new grammar definition. However, any node added to the parsing trees by the chunk operation has the probability of 1. Therefore, it does not affect $P(O|G)$, hence it does not affect the a posteriori change that a merge causes on the tree rooted at the new node added by chunk. In short, chunk does not change the production counts or operation probabilities, the latter can only be changed by the merge operation.

This concludes the proof that a chunk operation neither eliminates any existing potential merge operation nor changes the impact of any merge on the a posteriori probability of the grammar. ■

It should also be noted that merge may create new chunk opportunities. This is because replacing two different nonterminals with a single name may make two substrings that were previously different the same, thereby creating new opportunities for applying chunk.

F. Complexity Analysis

In this section we examine the complexity of the inference algorithm. Space complexity is simple, because the needed space is at all times proportional to the grammar size. Since each *chunk* and *merge* operation decreases the size of the grammar, the memory requirement is the most stringent at the initial phase when the initial grammar is proportional to the number of terminal symbols in the training data D. Hence, the space complexity is $O(D)$.

Since our inference step is $chunk^*\ merge$, the algorithm repeatedly performs searches for the best arguments of either a chunk or merge operation. Search for $chunk$ arguments entails evaluating the benefits of chunking each substring of length up to 5. It is accomplished by going through the grammar five times, each time storing the number of times the substring of length $1 \leq i \leq 5$ encountered repeated so far in the grammar, in a hash table. While best case of inserting an entry into this table takes $O(1)$ (assuming there are no collisions), the worst case is $O(log(l(G)))$ time where a binary search is done to find the entry in a chain. Therefore, the search takes $O(l(G)\log(l(G)))$ steps. After finding chunk's arguments, adjustment to the grammar takes $O(l(G))$ time. Hence, the time complexity of each chunk operation is $O(l(G)\log(l(G)))$ overall with $l(G) = O(D)$ decreasing over time.

Complexity of the $merge$ operation is defined by the number of steps necessary to find the best pair of nonterminals to merge. Denoting the number of nonterminals by n_t, we notice that n_t is defined by the operations performed. For each chunk operation it gets higher by one while with each merge operation it gets lower by one. The total number of chunk and merge operations cannot exceed D because each operation decreases the length of the grammar by at least one from its initial length of at most D. Likewise, for initial value of n_t we have $n_t \leq D + 1$ so taking into account that at most D merge operations can be performed, we have in general that n_t is $O(D)$. Denoting by c_j the number of clauses on the right hand side of the definition of nonterminal j, we have the obvious inequality:

$$\sum_{j=1}^{n_t} c_j < l(G). \tag{6}$$

Considering all pairs of nonterminals, we match each nonterminal with at most $n_t - 1$ others. So, according to the way we compute the impact of merge on a posteriori grammar probability, the work done checking all nonterminals versus all others is as follows:

$$\sum_{j=1}^{n_t} \left(c_j + \sum_{k \neq j} c_k \right) < \sum_{j=1}^{n_t} l(G) = n_t l(G). \tag{7}$$

We can further conclude that each $merge$ operation takes $O(n_t l(G)) = O(D^2)$ and this is also the complexity of each loop in Alg. 1.

Clearly, the number of loop repetitions in Alg. 1 cannot exceed the size of training data (D) because every $merge$ or $chunk$ operation decreases grammar size from its initial size of $O(D)$ at the sample incorporation stage. Hence, the worst-case time complexity can be expressed as:

$$T_W(D) = \sum_{i=1}^{D} O(D^2) = O(D^3) \tag{8}$$

where D denotes the total length of the training data measured in symbols used in its definition, rather than in the number of sentences.

G. Constrained Search for the Best Merge Arguments

In this section we show that if search for merge arguments is limited to nonterminals with a single rule, then the complexity of the merge search drops to $O(l(G))$. This restriction also lowers the complexity of the entire algorithm to $O(D^2 \log(D))$ because now $chunk$ becomes the most expensive action. This restriction is motivated by the fact that all nonterminals created by a chunk operation are of this form. Moreover, in our experiments we have seen that a nonterminal created by a $merge$ operation has two rules in most cases which indicates that this nonterminal was created by $merge$ of two nonterminals, each with a single production rule.

We now demonstrate why limiting the search for merge arguments as described above lowers the complexity of $merge\ search$ to $O(l(G))$. When two nonterminals each with a single rule (with counts n and m respectively) are merged, the a posteriori value improvement is:

$$P(G|O)_{new} = 2\ P(G|O)_{prev} \left(\frac{n}{n+m} \right)^n \left(\frac{m}{n+m} \right)^m \tag{9}$$

Indeed, probability values on parsing tree nodes are changed according to the last two terms and $l(G)$ is decreased by 1 (a nonterminal name is removed, as there is one nonterminal for both of the merged rules). If we remove the constant multipliers in the above equation, it can be seen that most advantageous merge couple (among nonterminals with a single rule) is the one that maximizes $\left(\frac{n}{n+m}\right)^n \left(\frac{m}{n+m}\right)^m$ where n and m are the rule counts of these two nonterminals.

We will next prove that if we have p counts (for nonterminal rules) $n_1 ... n_p$, then the smallest two of these will maximize the expression.

Theorem 2. *Let,*

$$F(x,y) = \left(\frac{x}{x+y} \right)^x \left(\frac{y}{x+y} \right)^y,$$

then value of F(n,m) increases if either n or m decreases.

Proof: We prove that if $m > 1$, then $F(n,m) < F(n,m-1)$. Let $R(n,m) = F(n,m)/F(n,m-1)$ then,

$$R(n,m) = \left(1 - \frac{1}{n+m}\right)^{n+m-1} \frac{m}{n+m} \left(1 + \frac{1}{m-1}\right)^{m-1}$$

$$= \left(1 - \frac{1}{n+m}\right)^{n} \frac{m}{n+m} \left(1 + \frac{n}{(n+m)(m-1)}\right)^{m-1}.$$

But $\left(1 + \frac{n}{(n+m)(m-1)}\right)^{m-1}$ is equal to

$$1 + \frac{n}{n+m} + \sum_{i=2}^{m-1} \binom{m-1}{i} \frac{n^i}{(n+m)^i} \frac{1}{(m-1)^i}$$

and then,

$$\sum_{i=2}^{m-1} \binom{m-1}{i} \frac{n^i}{(n+m)^i} \frac{1}{(m-1)^i}$$

$$< \sum_{i=2}^{m-1} \frac{(m-1)^i}{2^{i-1}} \frac{n^i}{(n+m)^i} \frac{1}{(m-1)^i} < \frac{n^2}{(n+m)^2}.$$

Finally,

$$R(n,m) < \left(1 - \frac{1}{n+m}\right)^n \left(1 - \frac{n}{n+m}\right) \left(1 + \frac{n}{n+m} + \frac{n^2}{(n+m)^2}\right)$$

$$< \left(1 - \frac{n}{n+m}\right) \left(1 + \frac{n}{n+m}\right) + \frac{n^2}{(n+m)^2} = 1.$$

It immediately follows that also $F(n,m) < F(n-1,m)$ by just repeating the argument above with n instead of m decreased by 1. ■

Hence, indeed the $merge$ operation with two smallest repetition counts increases the grammar a posteriori probability the most among all pairs. This reduces the $merge\ search$ problem into finding two nonterminals, each with a single rule, with the smallest rule counts, which can be done in $O(l(G))$ steps, which is also the complexity of $merge$ operation[1].

In the previous section we had proven that a $chunk$ operation has complexity $O(l(G)\log(l(G)))$. As chunk is now the most significant step in the loop of Algorithm 1 complexity-wise, the overall complexity of the algorithm becomes (we showed in the previous section that $l(G)$ is $O(D)$ and it decreases by at least 1 in chunk or merge operation):

[1]Recently, we have generalized the above result and showed that for a set of nonterminals, the most beneficial merge done on a pair of nonterminals from this set uses the nonterminals with the smallest sums of counts of rules. This result should enable us to improve the search merge complexity described above to arbitrary merge arguments.

$$T_w(D) = \sum_{i=1}^{D} D \log(D) = O(D^2\ log(D)). \quad (10)$$

V. Real-world Scenario and Simulations

In this section we give an example of how PCFGs can be used in a parking lot application where events of parking at a spot or leaving a parking spot can be detected. This is just a demonstration to show the usefulness of the approach.

The main problem in these kind of applications is what kind of sensor to use and how the terminal symbols should be chosen. A video camera can be used to monitor the parking area and by using object detection techniques, location of a car can be detected at any instance. From this information, we can set up the following simple terminal symbols:

- MOVE: Location changes
- STOP: Location does not change
- TURN: Direction changes (this is determined by looking at two previous locations of a car)

A grammar can be trained by using many examples for different possible events. Fig. 6 shows a car tracking example and tracking done to gather symbols from the camera image.

Fig. 6. Parking Lot Car Tracking for Event Detection

A simulation based on such a real-world scenario is given below. Parking lot is a 20x20 grid where any car can move one square at any time-unit. A car can go forward and backwards along its current direction, and it can change its direction by 45 degrees to the right or to the left of the current one. 90 degree turns are not allowed for a more realistic motion model. When a car enters the parking lot, the subsequent entries to the lot are delayed until either there is a new parking or exit from the parking lot made by a car (regardless if this is the most recently entered or some other car). We take one-square movement of any car to be one time unit and measured to this unit, parking (that is staying in parked state) time is exponentially distributed with mean of 5000 time units. The interarrival time of cars is also exponentially distributed but with mean of 5 time units. This does not congest the traffic in the parking lot because a car cannot enter the parking lot before the others park or leave it, and parking (or leaving) takes time. Cars that are not allowed to enter, simply queue up at the parking entrance.

Fig. 7 and Fig. 8 show the car trajectories associated with different events in the parking lot simulation. Squares with a circle in them represent the parking spots. We have assumed three events:

- Entering the parking lot, finding the closest available parking spot and then parking there (*Enter and Park*).
- Entering the parking lot, not being able to find an empty spot and leaving (*Enter and Leave*).
- Leaving the parking lot from a parking spot (*Leave Parking*).

First two of these events and their corresponding actions are shown in Fig. 7. Note that a car can circle around the lot for a few times to find a parking spot before leaving. In the simulations, we assumed that a car leaves the parking lot after an unsuccessful circle with probability of 60% and starts another circle around the parking lot with probability of 40%.

In Fig. 8, the movements of a car leaving the parking lot are shown. Please notice that a car goes first backwards from the parking spot to change its direction, which is often the case in real-life.

We have run the training data generation program (for feeding into grammar inference algorithm) with above given parameters for 100000 time units. Size of training data acquired for each event is: *Enter and Park (1468 sentences, 25333 symbols)*, *Leave Parking (1396 sentences, 17698 symbols)* and *Enter and Leave (952 sentences, 13565 symbols)*.

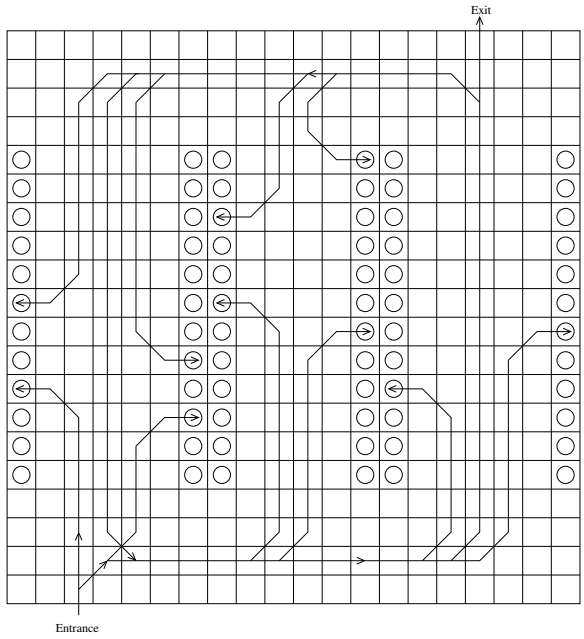

Fig. 7. Car trajectories for the events: *Enter and Park* and *Enter and Leave*

Fig. 9, Fig. 10 and Fig. 11 give the grammars inferenced from the results of our simulations. The values of interarrival and parking times dictate the probabilities of a single car being in one of the three events. Measured in our simulations, these probabilities are: *Enter and Park (38%)*, *Leave Parking (37%)*, and *Enter and Leave (25%)*.

Tokens (smallest action unit) for the grammars are: *en (enter)*, *ex (exit)*, *f (one or more forward actions)*, *b (one or*

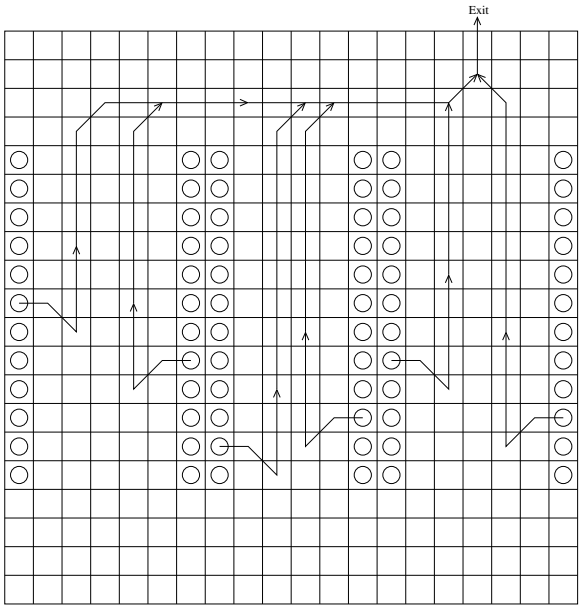

Fig. 8. Car Trajectories for the event *Leave Parking*

more backward actions), *tr (turn right 45 degrees)*, *tl (turn left 45 degrees)* and *s (stop)*.

START -->
M1 C3 C6 (0.004)
M1 C6 (0.062)
M1 C5 (0.063)
M1 C7 C6 (0.054)
N0 N1 N2 N1 N3 C2 C6 (0.134)
M1 C3 C5 (0.002)
M1 C7 C5 (0.012)
N0 C2 C5 (0.246)
N0 C2 C7 C6 (0.243)
N0 C1 C6 (0.130)
M1 C3 C7 C6 (0.003)
N0 C2 C7 C5 (0.047)

C7 -->
C1 C1 (1.0)

C5 -->
C1 C2 C6 (1.0)

C1 -->
N1 N3 N1 N3 (1.0)

N4 -->
s (1.0)

C6 -->
N1 N4 (1.0)

N1 -->
f (1.0)

N3 -->
tl (1.0)

N2 -->
tr (1.0)

C2 -->
N1 N2 N1 N2 (1.0)

C3 -->
C7 C7 (1.0)

N0 -->
en (1.0)

M1 -->
M1 C3 (0.151)
N0 C2 C3 (0.849)

Fig. 9. PCFG for event *Enter and Park*

The grammars fully model the actions that a car can do for certain events shown in Fig. 7 and Fig. 8. Moreover, the grammar for the event Enter and Park automatically discovers that the activity is recursive in the case when there are several circles around the parking lot before the spot is found. This is achieved by repeating *C3* in nonterminal *M1* (see Fig.9). If examined carefully, nonterminal *C3* reduces to *4*(f tl f tl)* which completes a circle around the parking lot (therefore the

START -->
N4 N2 N4 N2 N1 N2 N1 C1 N3 N1 N0 (0.333)
N4 N3 N4 N3 N1 N2 N1 C1 N3 N1 N0 (0.339)
N4 N2 N4 C1 N2 N1 N0 (0.172)
N4 N2 N4 N2 N1 C1 N0 (0.156)

N1 -->
f (1.0)

N4 -->
b (1.0)

N3 -->
tl (1.0)

N0 -->
ex (1.0)

N2 -->
tr (1.0)

C1 -->
N2 N1 N3 N1 (1.0)

Fig. 10. PCFG for event *Leave Parking*

START -->
C3 C4 (0.823)
C3 C2 C4 (0.152)
C3 C2 C2 C4 (0.024)
C3 C2 C2 C2 C2 C4 (0.001)

N1 -->
f (1.0)

N3 -->
tl (1.0)

N2 -->
tr (1.0)

N0 -->
en (1.0)

N4 -->
ex (1.0)

C1 -->
N1 N3 N1 N3 (1.0)

C4 -->
C1 N1 N4 (1.0)

C3 -->
N0 N1 N2 N1 N2 (1.0)

C2 -->
C1 C1 C1 C1 (1.0)

Fig. 11. PCFG for event *Enter and Leave*

grammar has recognized *a circle* as a sub-event). This also brings out the fact that grammar can produce more than the training data contain (generalization) simply by capturing the recursiveness. Input to the inference algorithm was a set of finite length strings while the output grammar produces strings of infinite length (however, probability of generating a string gets smaller as its length gets bigger). Please note that in the listed grammars, nonterminals with names starting with M (e.g $M1$) were created by $merge$ operations, while those whose names begin with C were created by $chunk$ operations.

It should also be noticed that in the grammar for the event Enter and Leave, the probability of circling again gets too small just after four circles around the parking lot to recognize potentially recursive circling in this case, as shown by the definition of **START** in Fig. 11.

An actual implementation of such an event recognition system could be done in two stages:

- *Training Stage*: Training data should be collected by a distributed set of sensors which, in our case could be camera sensors with tracking capabilities. Later, inference algorithm should be run at the base station while relatively small grammar definitions can be distributed to the sensor nodes. Clearly, the $O(D^3)$ worst-case algorithm complexity may require a larger processing power than the limited capability sensor nodes can provide today.
- *Recognition Stage*: After the grammar definitions are distributed to the sensors, all that a sensor has to do is to parse the input collected over the area that it is

monitoring. To keep this parsing more efficient, a sliding window or a progressive parsing (such as Earley-Stolcke [2], [9] parser) can be used which would hold different parsing possibilities and remove the ones that fail as the parsing progresses. Please note that parsing is a quite light task that can be done in $O(l^3)$ (Earley-Stolcke[2], [9], CYK[19], [20], [21]) where l is the length of the string to be parsed and thus much smaller than D.

Fig.12 presents a simple version of the deployment scheme proposed above.

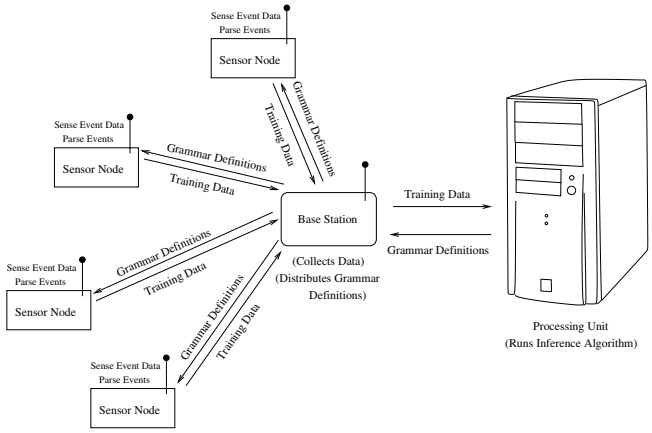

Fig. 12. Deployment Architecture for the Recognition System

VI. Conclusion and Future Work

We have presented a probabilistic context free grammar inference method for sensor network event recognition. Application of such a method prevents the problems caused by a manual construction of grammars. We have introduced a method for fast computation of Bayesian formula for evaluating a posteriori likelihood of a grammar. This method takes advantage of the properties of chunk and merge operations as well as the Bayesian posteriori likelihood formulation. We have also presented a real-world parking scenario where tracking input from a visual sensor can be interpreted as higher events of parking etc. We have implemented simulations of such a parking lot application and then generated grammars for events in these simulations. Those grammars demonstrate that the grammar inference method is able to catch the recursive nature of the parking action in such a scenario.

We are currently working on experiment settings that will provide us with the actual case for measuring effectiveness of this learning method for sensor network applications. We are also working on further improvement of the PCFG inference algorithm and its complexity.

Acknowledgment

This research was sponsored by US Army Research laboratory and the UK Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the authors, and should not be interpreted as representing the official policies, either expressed or implied, of the US Army Research Laboratory, the U.S. Government, the UK Ministry of Defense, or the UK Government. The US and UK Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

References

[1] Akyildiz, I. F., Su, W., Sankarasubramaniam, Y., Cayirci, E., *A Survey on Sensor Networks*, Computer Networks, 2002, pp. 393-422.

[2] A. Stolcke (1994), *Bayesian Learning of Probabilistic Language Models*, Doctoral dissertation, Dept. of Electrical Engineering and Computer Science, University of California at Berkeley.

[3] S. F. Chen (1996), *Building Probabilistic Models for Natural Language*, Doctoral dissertation, Dept. of Computer Science, Harvard University.

[4] Lymberopoulos, D., Ogale, A. S., Savvides, A., Aloimonos, Y., *A Sensory Grammar for Inferring Behaviors in Sensor Networks*, The Fifth International Conference on Information Processing in Sensor Networks, 2006. IPSN 2006.

[5] Teixeira, T., Culurciello, E., Park, J. H., Lymberopoulos, D., Savvides, A., Address-Event Imagers for Sensor Networks: Evaluation and Modeling, Proceedings of Information Processing in Sensor Networks, Platforms session, IPSN/SPOTS 2006.

[6] Teixeira, T., Lymberopoulos, D., Culurciello, E., Aloimonos, Y., Savvides, A., *A Lightweight Camera Sensor Network Operating on Symbolic Information*, Proceedings of First Workshop on Distributed Smart Cameras 2006, held in conjunction with ACM SenSys 2006.

[7] Lymberopoulos, D., Barton-Sweeney, A., Teixeira, T., Savvides, A., *An Easy to Program Sensor System for Parsing Human Activities*, ENALAB Technical Report 090601, May 2006.

[8] Ivanov, Y. A., Bobick, A. F., *Recognition of visual activities and interactions by stochastic parsing*, Pattern Analysis and Machine Intelligence, IEEE Transactions on, Volume 22, Issue 8, Aug. 2000, Page(s):852-872.

[9] Earley, J., *An efficient context-free parsing algorithm*, Communications of the ACM, Volume 13, Issue 2, Pages: 94-102, 1970.

[10] Moore, D., Essa, I., *Recognizing multitasked activities from video using stochastic context-free grammar*, In Proceedings of AAAI-02, 2002.

[11] Kitani, K. M., Sato, Y., Sugimoto, A., *Deleted Interpolation Using a Hierarchical Bayesian Grammar Network for Recognizing Human Activity*, Proceedings of the 14th International Conference on Computer Communications and Networks, Pages 239-246, 2005.

[12] Lari, K., Young, S. J., *The estimation of Stochastic Context-Free Grammars using the Inside-Outside algorithm*, Computer Speech and Language, 4(35-56), 1990.

[13] Baker, J., *Trainable grammars for speech recognition*, The Journal of the Acoustical Society of America, June 1979, Volume 65, Issue S1, p. S132.

[14] Stolcke, A., Omohundro, S., *Inducing Probabilistic Grammars by Bayesian Model Merging*, Proceedings of the Second International Colloquium on Grammatical Inference and Applications, Pages: 106-118, 1994.

[15] Jurafsky, D., Wooters, C., Segal, J., Stolcke, A., Fosler, E., Tajchaman, G., Morgan, N., *Using a stochastic context-free grammar as a language model for speech recognition*, International Conference on Acoustics, Speech, and Signal Processing, 1995. ICASSP-95. Volume 1, 9-12 May 1995 Page(s):189 - 192.

[16] Roark, B., *Probabilistic top-down parsing and language modeling*, Computational Linguistics, Volume 27, Issue 2 (June 2001), Pages:249 - 276.

[17] Sakakibara, Y., Brown, M., Hughey, R., Mian, I. S., Sjolander, K., Underwood, R. C., Haussler, D., *Stochastic Context-Free Grammars for tRNA Modeling*, Nucleic Acids Research, 22(23):5112–5120, 1994.

[18] Manning, C. D., Schutze, H., *Foundations of Statistical Natural Language Processing*, Cambridge, Massachusetts: MIT Press (1999).

[19] Cocke, J., Schwartz, J. T., *Programming languages and their compilers: Preliminary notes*, Technical report, Courant Institute of Mathematical Sciences, New York University, 1970.

[20] Kasami, T., *An efficient recognition and syntax-analysis algorithm for context-free languages*, Scientific report AFCRL-65-758, Air Force Cambridge Research Lab, Bedford, MA, 1965.

[21] Younger, D., H., *Recognition and parsing of context-free languages in time n^3*, Information and Control 10(2): 189208, 1967.