IEEE Transactions on Mobile Computing, vol. 12(11), Nov. 2013,pp. 2119-2131

Grammatical Inference for Modeling Mobility Patterns in Networks

Sahin Cem Geyik, *Student Member, IEEE*, Eyuphan Bulut, *Student Member, IEEE*, and Boleslaw K. Szymanski, *Fellow, IEEE*

Abstract—Modeling of the mobility patterns arising in computer networks requires a compact and faithful representation of the mobility data collected from observations and measurements of the relevant network applications. This data can range from the information on the mobility of the agents that are being monitored by a wireless network to mobility information of nodes in mobile network applications. In this paper, we examine the use of probabilistic context free grammars as the modeling framework for such data. We present a fast algorithm for deriving a concise probabilistic context free grammar from the given training data. The algorithm uses an evaluation metric based on Bayesian formula for maximizing grammar *a posteriori* probability given the training data. We describe the application of this algorithm in two mobility modeling domains: (i) recognizing mobility patterns of monitored agents in different event datasets collected by sensor networks, and (ii) modeling and generating node movements in mobile networks. We also discuss the model's performance in simulations utilizing both synthetic and real world mobility traces.

Index Terms—PCFG, Computer Networks, Event Recognition, Mobility Modeling, Data Generation.

1 Introduction

We define data modeling as the task of finding a model of the given data that is compact and includes enough features to construct samples synthetically from it with the same statistical properties as the input. Such a model has the following advantages:

- Saving disk-space or reducing data transmission cost in constrained systems;
- Supporting recognition of data confirming to the model, i.e., the ability to detect if a certain sequence of data belongs to the model or not;
- Predicting the data that will be produced by an environment;
- Enabling generation of large amounts of data that possess properties of the original data and therefore can be utilized for testing purposes, etc.;
- Facilitating the manual inspection of data properties, thanks to a concise model description.

This paper focuses on modeling of the mobility data in network applications by utilizing probabilistic context free grammars (PCFGs). Informally, PCFGs are regular context free grammars with probabilities assigned to nonterminal productions. Our motivation to use this fairly complicated model is to be able to accurately capture human mobility. Usually, humans move according to a plan: sometimes simple, when the last location defines possible next locations, or sometimes more complex, when the next feasible locations depend on variable depth past, or the times of passages. Some plans may involve palindromic movements, or recursive movements repeating with decreasing probability etc. Some of the most sophisticated models of the past, based on Markov Models, do not consider variable length past. A simple example could be a parent dropping a child at nursery on the way to work, and picking the child on the way back home. Here, visits to a nursery are followed predictably by a visit to home or work, depending what the two previous locations were. PCFGs through use of production rules encode in them the smallest needed but still unbounded depth of past necessary to infer the possible next locations. Even for methods that store the variable length history from the training data, the modeling capabilities are theoretically bounded to mobility patterns which follow a regular language, while, as presented in this work, this is not the case for PCFGs. Finally, PCFG inference rules enable for powerful generalizations, which encode patterns not present in the data but constituting recursive closure of them. An example that we discuss in the paper is the circling over a parking lot in a search for a free parking spot. All these and more examples motivate our choice of PCFGs as a sophisticated mobility model with capabilities far beyond others that is still computationally feasible for meaningful input data.

This paper makes the following contributions:

- A customized inference method for PCFGs which works directly on training data containing mobility patterns of interest. This algorithm improves and simplifies the two previous inference methods proposed in [33] and [34].
- Two mobility-specific extensions on top of the basic PCFG methodology. Time tokens are added for temporal modeling, and the relative tokens are introduced to facilitate generalization and to lower the complexity of the grammar creation.
- Demonstration of PCFG-based mobility modeling in two domains: (i) mobility pattern extraction in sensor networks, and (ii) synthetic trace generation in mobile networks.

The rest of the paper is organized as follows. First, we provide a review of previous work in mobility mod-

- *Sahin Cem Geyik, Eyuphan Bulut and Boleslaw K. Szymanski are with the Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY, 12180. E-mail: {geyiks,bulute,szymansk}@cs.rpi.edu*

eling. In the following section, we define probabilistic context free grammars which form the foundation of our model. Next, in Section 4, we describe our grammar construction algorithm which is an improvement over our two previous publications [1][2]. Section 5 presents our extensions to the basic PCFG definition to increase PCFG usability in mobile computing and networking. Some initial applications of this kind were discussed in [1] and [2]. We demonstrate the benefits of PCFG-based mobility modeling compared to other models in terms of describing entity movements in Section 6. In Section 7, we evaluate the PCFG framework by demonstrating its use in two different mobility modeling applications. Section 8 outlines our conclusions.

2 Previous Work

Mobility modeling is at the core of mobile computing research. It is needed to generate test data (synthetic trace generation), predict the future locations of nodes for different applications (e.g. connectivity prediction, data dissemination, target tracking etc.) and various other purposes. Initial efforts in modeling mobility have focused on random movement of nodes: e.g. random-waypoint [3], Manhattan [4] etc., even though real domains often contain obstacles or preset paths. Recent works that address these challenges include [5], [6], [7], and [8]. In [5], anchor points are used to describe how a mobile entity can move in an environment with obstacles. [6] introduces *Social Manhattan Mobility Model* where the original Manhattan Mobility Model is supplemented with additional social attraction points. Heterogeneous Community-based Random Way-Point (HC-RWP) mobility model is presented in [7]. It takes into account the locations visited and the movement preferences of different mobile nodes. A similar work is presented in [8] where the authors extend the random-waypoint model is enhanced with an *Attractor Matrix* that represents the similarity of social attributes of nodes.

There have also been many attempts at creating synthetic mobility patterns based on traditional random movement mobility models, including methods based on connectivity graphs [9], action profiles [10], terrain and vehicle properties [11], group behavior [12] and events [13].

While the methods based on modeling randomized movements are useful due to their mathematical tractability and simplicity of generating synthetic traces, they suffer from their inability to closely represent realistic movements. To address this shortcoming, a set of efforts have been published that work on real world traces collected at various settings [14][15]. A time-variant community mobility model is proposed in [16]. The model utilizes both skewed location visiting preferences and the periodical reappearance at the given location to generate mobility traces. Urban pedestrian flows (UPF) based mobility scenarios are discussed in [17]. They generate mobility traces based on pedestrian densities on streets as well as likely paths that the pedestrians take. Both [16] and [17] make use of real world traces to build their models for generating synthetic traces. In [18], a unified relationship matrix is used to describe social strengths between people within the same community or in different communities. This in turn determines the colocation of people. Finally, [19] introduces a graph based mobility model, in which the vertices of the graph represent geographical locations and the edges represent paths between those locations. Certain other properties of movement, such as speed, are kept as parameters within the vertices.

The works closest to ours utilize Markov Models. In [20], transitions between areas are modeled by their probabilities. Another work is given in [21], which predicts AP associations of mobile users due to their up to two previous states (i.e. AP associations), hence utilizing a first or second order Markov chain. Markov Model based mobility predictors are compared to LZ-based [22] mobility predictors in [23] and the results show that Markov Models perform better. Interestingly, the paper also demonstrates that in practice, a 2-level Markov Model predictor performs better than a 3-level or 4-level predictor, hence increasing the depth does not necessarily increase prediction accuracy. Markov Models were extended by adding time information through cumulative time distribution of transitions in [24]. A similar work by the same research group presents Model T++ [25], a joint time-space access point (AP) registration model. In this work, the authors model distinct sets of APs as clusters, whereas the probabilities of transitions between APs in the same or different clusters are inferred from the training data. Furthermore, the association length of mobile entities to an AP before moving to another one (transition) is modeled by a Weibull distribution.

3 Probabilistic Context Free Grammars (PCFG)

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 (referred to also as nonterminals);
- S_t is a list of terminal symbols;
- *Start* is the sentential nonterminal where each sentence that the PCFG can produce generates from;
- *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 probability of using the associated production rule versus any other rule defining the same nonterminal.

The following simple PCFG[1] outputs strings of the form $a\ b^*$ with an average length of 1.75 symbols:

$$START \rightarrow a\ B\ (0.6) \mid a\ (0.4)$$

$$B \rightarrow b\ (0.8) \mid b\ B\ (0.2)$$

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 the above grammar, the string *a b b* has the probability of being produced equal to P(Start → a B) · P(B → b B) · P(B → b) $= 0.6\ \cdot\ 0.2\ \cdot\ 0.2\ \cdot\ 0.8 = 0.0192$.

Probabilistic Context Free Grammars have many uses in speech recognition [26], natural language processing [27], computational biology [28], sensor networks [29] [30] etc.

4 Grammar Inference

There are two different tasks associated with the PCFG inference problem: (i) setting the probabilities of the already given rules, and (ii) constructing the whole grammar from training data. For the first case, the well known so-called inside-outside algorithm [31][32] serves as the solution. For the second case, two previous papers, which also form the basis for our algorithm, are relevant. The first one, [33], uses the two operators (*merge* and *chunk*) as well as initial construction method which we used in our algorithm. Although we utilize a similar Bayesian formulation to evaluate the grammar, 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 work, [34], follows an opposite approach, and begins with a very general grammar. Later, this grammar is made specific to the training data with five operations (*concatenation*, *classing*, *repetition*, *smoothing* and *specialization*).

In this section, we explain how our PCFG inference scheme works step by step. We also define the operators utilized as well as the scoring scheme for the goodness of the inferred grammar.

4.1 Operations Used for Grammar Construction

The grammar inference method that we apply to network mobility data modeling uses the same steps that were introduced by Stolcke [33]. Grammar construction consists of two phases: sample incorporation, and application of operators.

Sample Incorporation: This stage constructs an initial grammar from the training data ($D = \{d_1, d_2, \ldots\}$). Each sentence (d_k) in training data is a string of terminal symbols ($symbol_i$), which are introduced into the grammar as nonterminals by productions of the form $N_i \rightarrow symbol_i$ with frequencies defined by the training data. The goal is to separate $terminal$ productions from $nonterminal$ productions.

The sentences are introduced to the initial grammar by rules of the form $START \rightarrow N_{s,1}\ ...\ N_{s,j}$ where each nonterminal symbol in such a rule corresponds to a single terminal from the training data appearing in the training data sequence. The frequency of each production is equal to the frequency of the right sentence represented by the rule in the training data. Later, the *maximum likelihood* estimate of any rule's probability can be calculated as:

$$P(rule_i) = \frac{frequency(rule_i)}{\sum_{k=1}^{n} frequency(rule_k)}\ , \qquad (1)$$

where n is the number of rules in the definition of the nonterminal to which that rule$_i$ belongs.

Operators: Two operators: merge and chunk [33], are used to build the grammar step by step. Each operator has a different effect on the grammar. We explain how these two operators work and how they affect the grammar by examples.

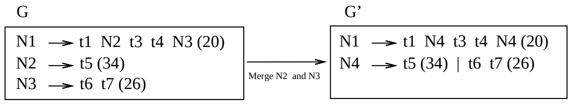


Fig. 1. An Example of Merge Operation

Merge takes two nonterminals and reduces them into a new nonterminal by combining their rules, as well as the frequencies. See Figure 1 for an example, where nonterminals *N2* and *N3* are first combined into a new nonterminal *N4*, and then they are removed by replacing all of their occurrences in the grammar with *N4*. The parentheses in the figure contain the rule frequencies.

Merge is a generalizing operator as the resulting grammar accepts sentences that do not exist in the training data. It can also create recursion if combining two nonterminals *X1* and *X2* when the rule *X1* → ... *X2* exists. This is because in such case the new nonterminal (let's say X_{new}) will contain the rule $X_{\text{new}} \rightarrow \ldots X_{\text{new}}$.

Two special cases might occur during merge, first one when a nonterminal is merged with $START$ nonterminal. In such case, the new nonterminal is also called $START$. The second case happens when one of the nonterminals include the other one as a rule with a single symbol. In this case that rule is simply removed from the transformed grammar, since its replacement would be meaningless. For example, if *X1* includes the rule *X1* → *X2* (while merging *X1* and *X2*), then the new nonterminal (*X3*) would contain the rule *X3* → *X3*, which is removed together with its frequency.

Chunk operator creates a new nonterminal with a single rule which is a string composed of symbols in

1. In this paper, we start nonterminal names with an upper-case and terminal names with a lower-case letter for presentational purposes.

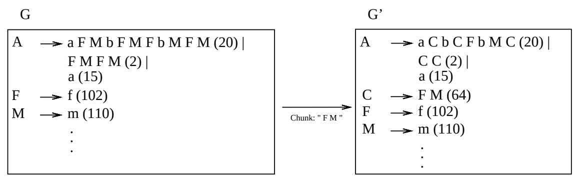


Fig. 2. An Example of Chunk Operation

the current grammar. Each occurrence of this string is replaced with the new nonterminal. The frequency of this nonterminal is equal to the total number of replacements multiplied by the frequency of the rules in which the replacement takes place. Hence this frequency can be interpreted as the number of times the corresponding pattern (that is, the single rule in the new nonterminal) exists in the training data. In Figure 2, the pattern *F M* defines the new nonterminal *C*, and the frequency is set according to the number of replacements of these patterns by *C* in all the rules.

4.2 Evaluation Metric for the PCFG

It is a crucial task to evaluate the *goodness* of the grammar (G) that is constructed from the training data (D). For this purpose we utilize the Bayesian *a posteriori* probability ($P(G|D)$) [35]:

$$P(G|D) = \frac{P(G)P(D|G)}{P(D)} \ . \tag{2}$$

The prior for the training data ($P(D)$) can be removed from the above formulation; this way, the evaluation metric becomes $P(G)P(D|G)$. Here, $P(G)$ is the grammar *a priori* probability which, based on *Occam's Razor* principle, is inversely related to the grammar description length (denoted as l_G). From information theory, we have the following equation for $P(G)$ (α is used as a coefficient to represent the space of possible grammars, and does not affect grammar inference process):

$$P(G) = \alpha \ \cdot \ 2^{-l_G}. \tag{3}$$

$P(D|G)$ stands for the likelihood of the training data, given the grammar G and calculated by multiplying the probabilities of all the sentences in D:

$$P(D|G) = \prod_{i=1}^{|D|} p(d_i|G). \tag{4}$$

We employ a simple constant bit length (l_s for each symbol) representation of the grammar to calculate l_G. For each nonterminal in the system, l_G increases by this length (corresponding to the nonterminal name). For each rule in the nonterminal, l_G increases by $l_s + ((number\ of\ symbols\ in\ the\ rule)\ .\ l_s)$, which also accounts for the separation symbol. Different representation schemes can be employed, but the advantage of our representation is that, as shown later, it limits the scope of search for the operands to the chunk operation.

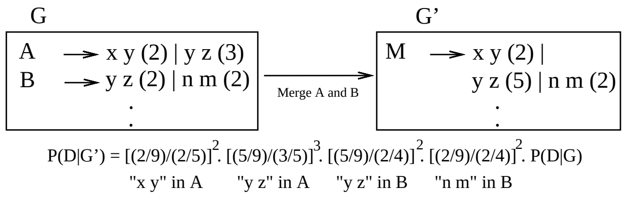


Fig. 3. Calculation of $P(D|G)$ for Merge Operation

Our goal via construction is to find a grammar that is generalized however also keeping above a certain *a posteriori* probability. As we have previously mentioned, the initial grammar we consider is created by the *sample incorporation* stage. This grammar generates only the sentences from the training data D, and is the grammar that has the *maximum likelihood* for the training data ($P(D|G)$). This grammar is however too specific, and also of largest description length. What we would like to achieve is to shorten this grammar (by the *chunk* operator which will decrease the size and increase *a posteriori* probability) and make it more generalized (by the *merge* operator which will decrease the likelihood and hence the *a posteriori* probability). Of course the generality of the grammar should be bounded, e.g. a grammar that accepts all strings is *overgeneralized* and is not a good choice for any application domain. In our work, we tried to achieve the balance between the generality and the specificity by keeping the *a posteriori* probability always above or equal to the initial grammar's (via the sample incorporation stage) *a posteriori* probability. The details of how we achieve this is given in Section 4.5. This choice is further justified by good results from applications presented in Section 7.

4.3 Computation of the Evaluation Metric

In this section, we describe the methods that we use to calculate the effect of the operators (Section 4.1) on the *a posteriori* probability (Eq. 2).

4.3.1 Chunk

For any chunk operation, the change in $P(G)$ is easy to calculate via the modified grammar, furthermore $P(D|G)$ does not change. We demonstrate these properties on the example presented in Figure 2. When *F M* is chosen as the chunk nonterminal, each sentence production that uses a rule that is modified by this chunk now goes to the new rule (consisting of *F M*) in its parsing tree, but its probability does not change since this new rule has probability of 1.0. As an example, if a sentence uses $A \rightarrow F\ M\ F\ M$ (which has 2/37 probability), it now uses $A \rightarrow C\ C$ and twice $C \rightarrow F\ M$ which still has 2/37 probability in total.

4.3.2 Merge

Calculating $P(G')$ is again easy, given the modified grammar G'. However, $P(D|G')$ changes whenever merge operation takes place during modification of the grammar from G to G'. A naive and inefficient approach would be to re-parse the training data to compute $P(D|G')$ anew. However, if we consider the ratio $P(D|G')/P(D|G)$, probabilities of all unchanged rules will cancel out, leaving only modified rules, so we can look just at the change in the estimated probabilities of the modified rules and how frequently these rules are used.

As shown on example from Figure 3, the frequency of a rule is used as a power to the change in the rule's probability. This applies to all rules that have a probability change, in other nonterminals as well, since a merge of two nonterminals can affect a third nonterminal (e.g., rules becoming the same due to replacements).

4.4 Search for the Best Merge and Chunk Arguments

As discussed in Section 4.3.2, since a merge operator can affect many nonterminals at once, all pairs of nonterminals should be examined for merge, and the pair that gives the highest *a posteriori* probability will be chosen. Approximations of this method to lower time complexity are discussed in Section 4.7.

Search for the best chunk argument rapidly becomes expensive with the growth of grammar size if we check strings of all lengths as arguments. By using the fixed bit length symbol representation, the length of the strings that need to be considered can be bounded from above by length of 5. Indeed, chunking a string which occurs just once cannot shrink the grammar, and at each replacement of a string of length k with the chunk nonterminal, the grammar gets $k-1$ symbols shorter. However, the chunk nonterminal itself adds $k+2$ to the grammar length (k symbols + nonterminal name + separation symbol). This should be made up for by frequent replacements, and let's say there are n replacements of the chunk string in the grammar with the chunk nonterminal. Then we must have $n(k-1) > k+2$ for a chunk to be advantageous. For any chunk, $n \geq 2$, so in the worst case of a chunk string occurring twice, the length of the string must be at least 5 ($k > 4$). Of course if there are no strings up to length 5 that occurs at least twice, then there are no advantageous chunks. If there are chunks longer than 5 and appears twice or more, these can be discovered by further chunk argument searches once the previous chunk operator is applied. For instance, if the repeating string is longer than 8, then its subsequent part will still be chunked, so by using this method we lose only a little in terms of lost chunk opportunities (potentially strings of length $5k+1$, $5k+2$, $5k+3$, for $k = 1, 2, 3$ ...).

Size difference between two grammars (G before and G' after chunk operation) is expressed by the following formula:

$$l_G - l_{G'} = l_s[(n-1)(k-1) - 3]. \qquad (5)$$

4.5 Inference Algorithm

In most cases, merge operation decreases *a posteriori* probability according to definition of $P(G|D)$ given by Eq. 2. When, during merge, we replace a production with a small number of alternatives by the one with a larger number of alternatives, the advantage of the shorter grammar length usually does not compensate for the decrease in $P(D|G)$. 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 inference steps is $chunk^*\ merge$, where we do all the beneficial chunk operations before performing any merge operation.

Algorithm 1 PCFG Inference Algorithm

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

The outline for this scheme is given in Algorithm 1. As it can be seen, rather than trying to always find advantageous merge and chunk operators, we try to find a $chunk^*\ merge$ step that keeps the *a posteriori* probability above the one that the initial grammar (via the *sample incorporation* stage) had. If the algorithm is carefully examined, we give the *a posteriori* probability of β (a random starting value since we are interested in relative *a posteriori* differences between grammars) to the initial grammar, and we keep the grammar *a posteriori* probability always above this value (hence in some sense *at least as good as* the initial grammar). Each chunk operation increases the *a posteriori* probability, and even if the most advantageous merge operator that is found tends to decrease the *a posteriori* probability, it is applied if the value is still above β, hence allowing for generalization. This is thanks to the *cushion* that is provided by the applied chunk operations. Also, as presented in the next section, Algorithm 1 terminates due to the fact that each operator shrinks the grammar, hence the *while* loop is repeated at most D times (D being the size of training data). Please note that different strategies can be applied, which may allow for further generalization, or keep *a posteriori* probability always at higher values. Particularly, in our work, rather than utilizing only the final grammar (i.e. the grammar when Algorithm 1 terminates), we have also utilized the grammar which had the highest *a posteriori* probability during the construction process.

In our previous work [1], we prove that any chunk operation neither eliminates feasible merge operations (actually may add to them) nor changes their impact on *a posteriori* probability of the grammar. We omitted this proof here due to limited space, but interested readers can refer to [1]. This proof furthermore justifies our $chunk^{*}\ merge$ choice, which is a distinct diversion from [33], where the step can be seen as $chunk\ merge^{*}$.

4.6 Complexity Analysis

In this section, we examine the complexity of the inference algorithm. First of all, space complexity is $O(D)$ (we represent the size of the training data by D as well). In sample incorporation stage, the initial grammar size is D, and each operator later decreases the size of the grammar, hence the space requirement is always below or equal to D.

Based on our inference step ($chunk^{*}\ merge$), our algorithm tries to find best arguments for a chunk or a merge operator. A straightforward implementation of chunk argument search basically requires going through the grammar l_{max} times, where l_{max} is the longest chunk that we search for. If it is taken to be the longest rule (which finds the optimal chunk), then a chunk argument search takes $O(l_G \cdot l_{max} \cdot \log(l_G))$, since to count the frequencies of strings, we need a hash table with a worst-case time complexity of $O(\log(l_G))$ for inserting or finding (to update frequency) a string. This furthermore can be taken as $O(l_G^2 \log(l_G))$ since it is safe to say l_{max} is in the order of $O(l_G)$. Our improvement over [33] is to look in rules for repeating strings of length of at most 5, which finds all advantageous chunk operands, if they exist. This significantly reduces[2] the complexity of the search for a chunk to $O(l_G \log(l_G))$. After finding the chunk string, the modification of the grammar takes $O(l_G)$, hence the chunk operator takes an overall $O(l_G \log(l_G))$ time 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. It increases by one after each chunk operation and decreases by one after each merge operation. 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 $O(D)$. Considering all pairs of nonterminals, we match each nonterminal with at most $n_t - 1$ others, and since each check takes $l_G = O(D)$ (by applying the merge operator and finding the change in *a posteriori* probability), we can conclude that each $merge$ operation takes $O(n_t^2 l_G) = O(D^3)$ 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 $T_W(D) = \sum\limits_{i=1}^{D} O(D^3) = O(D^4)$, where D denotes the total length of the training data measured in number of symbols.

4.7 Constrained Search for the Merge Arguments

In this section, we will show our two approximations on the merge search, which will lower time complexity of the inference algorithm and are improvements over [33]. First approximation we propose is to only look at the descriptions of the nonterminals that are being merged, hence ignoring the merge's effect on the rest of the grammar. Let's examine the complexity of this approach. Denoting by c_j the number of clauses on the right hand side of the definition of nonterminal j, we have the obvious inequality $\sum_{j=1}^{n_t} c_j < l_G$. Considering all pairs of nonterminals for a merge, we match each nonterminal with at most $n_t - 1$ others. In this case, since we only look at the right hand sides of the nonterminals that are being merged (and not the whole grammar as the previously shown exact approach), the time complexity of checking all nonterminal pairs (hence the merge search) takes $\sum_{j=1}^{n_t} \left(c_j + \sum_{k=j+1}^{n_t} c_k \right) < \sum_{j=1}^{n_t} l_G = n_t l_G$. By following this scheme, the merge operation is $O(D^2)$ (since $n_t = O(D)$ and $l_G = O(D)$; furthermore, applying the merge takes $O(D)$), and the inference algorithm overall takes $O(D^3)$, due to at most D loops.

The second approximation builds on the first one as follows. When checking the right hand sides of only the merged nonterminals, if we ignore the rules being deleted (due to being the same or being equal to the new merge nonterminal), then choosing the two nonterminals with the smallest total frequency gives the merge with the highest advantage (or the least disadvantage) on the *a posteriori* probability of the grammar.

Suppose that we have two nonterminals X and Y with rule frequencies $x_{1\to n}$ and $y_{1\to m}$ respectively (X has n rules and Y has m rules). Let's also denote the sums of frequencies as $\sum_1^n x_i = S_X$ and $\sum_1^m y_i = S_Y$. Then, the merge of these rules $M_{X,Y}$ has these rules

2. Although not used for implementation simplicity, the search for a chunk that maximizes the benefit function $f = [(n-1)(k-1)-3]$ can be done in $O(D\log(D))$ by finding all non-overlapping repeating chunks in the grammar via building a minimal augmented suffix tree (MAST) [36]. Hence, with the same complexity of $O(D\log(D))$, we can achieve the largest reduction of the grammar size with a single chunk operation.

side by side, making the change in the *a posteriori* probability as follows:

$$P(G|O)_{new} = 2^{l_s} \cdot P(G|O)_{prev} \cdot \left(\frac{x_1/[S_X+S_Y]}{x_1/S_X}\right)^{x_1} \cdot \ldots \cdot \left(\frac{x_n/[S_X+S_Y]}{x_n/S_X}\right)^{x_n} \cdot \left(\frac{y_1/[S_X+S_Y]}{y_1/S_Y}\right)^{y_1} \cdot \ldots \cdot \left(\frac{y_m/[S_X+S_Y]}{y_m/S_Y}\right)^{y_m}$$
$$= 2^{l_s} \cdot P(G|O)_{prev} \cdot \left(\frac{S_X}{S_X+S_Y}\right)^{S_X} \cdot \left(\frac{S_Y}{S_X+S_Y}\right)^{S_Y} .$$

We will next prove that to maximize $\left(\frac{S_X}{S_X+S_Y}\right)^{S_X} \cdot \left(\frac{S_Y}{S_X+S_Y}\right)^{S_Y}$ (and get the most advantageous *a posteriori* probability change), S_X and S_Y must be chosen as small as possible.

Theorem 1. *Let,* $F(n,m) = \left(\frac{n}{n+m}\right)^n \left(\frac{m}{n+m}\right)^m$, *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. □

This reduces the *merge search* problem to finding two nonterminals with the smallest total rule frequencies, which can be done in $O(l_G)$ steps, which is also the complexity of *merge* operation.

In the previous section we had proven that a *chunk* operation has complexity $O(l_G \log(l_G))$. As chunk operator 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) $T_w(D) = \sum_{i=1}^{D} D\log(D) = O(D^2 \ \log(D))$.

4.8 Summary of Our Contributions over Previous Grammar Inference Work

As aforementioned, our grammatical inference builds on previous work [33][34] by adopting the operators and the evaluation metric used previously. Although we have presented the grammar inference already, here we would like to detail our improvements:

- A simple representation of the grammar for calculating description length. This limits the search for the chunk operator arguments, reducing the time complexity of this operation from $O(D^2\log(D))$ to $O(D\log(D))$.
- A new approach to grammar construction process (i.e. keep *a posteriori* above the initial value) where the basic inference step becomes *chunk* merge*.
- A detailed analysis of the time complexity of the grammatical inference method.
- Two approximations to searching the merge operator arguments, which decrease the time complexity of the overall algorithm first from $O(D^4)$ to $O(D^3)$ and then to $O(D^2\log(D))$ (making now chunking the more costly operator).

5 Extensions to the PCFG

In this section, we present our extensions to the PCFGs to make them more efficient in mobility data modeling. First, we introduce time tokens, which represent the temporal properties of the application domain. The second one is the addition of relative tokens, which change the state of a modeled object according to its last state.

5.1 Time Tokens

We present a special time terminal symbol, t, to represent time that passes between occurrence of two terminals, each of which may represent an action or a location that the mobile node is in. In our definition, t represents a preset time length, and the number of subsequent t's determines the amounts of time represented by such a sequence. Suppose that we have three actions A, B and C happening consecutively in an application domain. Then $''A \ \ 24 \ \ B \ \ 42 \ \ C''$ means that 24 time units pass between A and B, and 42 time units pass between B and C. If we take a time token (t) to be 20 seconds, then this sequence can be represented approximately by $''A \ \ t \ \ B \ \ t \ \ t \ \ C''$. Please note that there is a trade-off between accuracy and complexity, since if we had taken for example the time token to be 6 seconds, then the amount of time passed in each case would be perfectly captured, but this would increase the number of symbols in the sentence, making inference processing more difficult. Time tokens help with temporal property representation in a grammar, and is of vital importance in mobility modeling.

5.2 Relative Tokens

Another extension we list here provides the state difference in a sequence (e.g. *movement* for mobility) and is called relative tokens. See Fig. 4 for an example. By introducing U (up), D (down), R (right), and UR (up-right) terminals, we can get rid of having every location name as a separate terminal (only the ones that a movement sequence starts from should be terminals), which reduces the complexity of the inference process. Furthermore, we can more easily detect frequent patterns. Again in Figure 4, it is easy to define a zig-zag motion as $UR\ DR$ (up-right and down-right, i.e. $\nearrow\searrow$). If we represent the exact locations in sentences, the corresponding motions will not be frequent; while in the case of relative tokens, a zig-zag subaction will be found as a chunk nonterminal no matter what the exact locations of zig-zaging objects are.

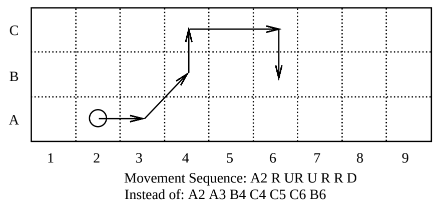

C
B
A
1 2 3 4 5 6 7 8 9
Movement Sequence: A2 R UR U R R D
Instead of: A2 A3 B4 C4 C5 C6 B6

Fig. 4. Relative Movement Token Example

6 Advantage of Utilizing PCFGs over Previous Models

In this section, we first describe how traditional mobility models can be described via a PCFG. Then we present our motivation for using PCFGs, and show its benefits.

6.1 Expressing Other Mobility Models as PCFGs

There are many previous mobility models that can be expressed in a PCFG-based description. For demonstration purposes, we will show transformations of the random-waypoint model and a Markovian mobility model to a PCFG description.

6.1.1 Random-Waypoint to PCFG Transformation

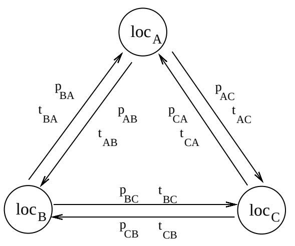
A mobile node adhering to the random-waypoint mobility model [3] moves in steps. In each step, it chooses a random point in the mobility area and then gets there with a random speed. For this type of randomized movement, we are using two randomized nonterminals: Loc_A and $V_{[f,c]}$. Loc_A basically instructs the grammar to generate a random location within the area A, while $V_{[f,c]}$ instructs the grammar to generate a random speed between the values floor (f) and the ceiling (c). The continuous location and speed generation is performed iteratively. In this setting, the random-waypoint grammar can be represented by the following PCFG:

$$\text{Start} \rightarrow \text{Loc}_A\ \text{V}_{[f,c]}\ \text{Start}\ (\text{p}_1) \mid \text{Loc}_A\ (\text{p}_2)$$

$$\text{Loc}_A \rightarrow \text{Random Location with a Preset Distribution}$$

$$\text{V}_{[f,c]} \rightarrow \text{Random Speed with a Preset Distribution}$$

Thus, at each *Start* production, the above grammar decides on a new location and the corresponding node moves there with the random speed generated in the previous production. The first production of the nonterminal determines the initial location of the mobile node. The length of the mobility period for a node is defined by the choice of probability p_1 (since $p_2 = 1 - p_1$), which determines how many times the relocation is expected to occur.



Transformation to PCFG

$\text{Start} \rightarrow \text{Loc}_A (p_A) \mid \text{Loc}_B (p_B) \mid \text{Loc}_C (p_C)$

$\text{Loc}_A \rightarrow \text{loc}_A\ T_{AB}\ \text{Loc}_B (p_{AB}) \mid \text{loc}_A\ T_{AC}\ \text{Loc}_C (p_{AC})$

$\text{Loc}_B \rightarrow \text{loc}_B\ T_{BA}\ \text{Loc}_A (p_{BA}) \mid \text{loc}_B\ T_{BC}\ \text{Loc}_C (p_{BC})$

$\text{Loc}_C \rightarrow \text{loc}_C\ T_{CA}\ \text{Loc}_A (p_{CA}) \mid \text{loc}_C\ T_{CB}\ \text{Loc}_B (p_{CB})$

$T_{AB} \rightarrow$ Time amount following distribution t_{AB}
$T_{AC} \rightarrow$ Time amount following distribution t_{AC}
$T_{BA} \rightarrow$ Time amount following distribution t_{BA}
$T_{BC} \rightarrow$ Time amount following distribution t_{BC}
$T_{CA} \rightarrow$ Time amount following distribution t_{CA}
$T_{CB} \rightarrow$ Time amount following distribution t_{CB}

Fig. 5. A Markovian Mobility Model and Its PCFG Transformation

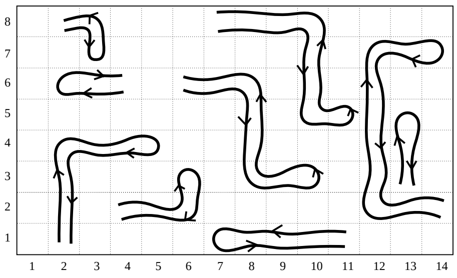
6.1.2 Markovian Model to PCFG Transformation

Markov model based mobility models [20][21][23][24] set a probability for the next location given the previous location(s) as well as the time that passes between two visits. We provide a similar mobility model in Figure 5 which has three locations (presented as states: loc_A, loc_B and loc_C) and movement (transition) probabilities between those locations (e.g. p_{AB} is the probability of moving from location loc_A to loc_B) with the distribution of time that passes for each transition (e.g. t_{AB} is the distribution of time that a node takes to move from location loc_A to loc_B). In the same figure, we provide the transformation of such a Markovian mobility model to the PCFG representation. In this case, we create a new nonterminal for each location that produces two terminals. The first terminal represents the location. The second terminal represents the time with the given distribution for transition of the node from this location to the next. The transition is represented as a triple of: terminal of the start location, terminal representing the transition time distribution, and terminal representing the end location. The *Start* nonterminal production in this case lists all possible initial locations for each node with steady state probabilities (e.g. p_A is the probability of a node being at loc_A in steady state) assigned to each alternative. Very similar transformations can be made for any n-level Markovian mobility model (in which the next location depends on previous n locations). These include Sample Pattern Matching (SPM) [37], Prediction by Partial Matching (PPM) [38], and LZ-Based [22][39] approaches which in practice may utilize a variable-length context.

6.2 Benefits of PCFG-based Modeling

We have demonstrated that the PCFGs are sufficient to represent many previously proposed methods for mobility modeling. Here we will present the benefits of using PCFGs both in terms of capturing real world traces, as well as in terms of expressing certain theoretically distinct mobility properties.

The first advantage of PCFGs is their ability to model time dependencies of variable nature. Let's take a Markov model as an example. Depending on the level it supports, a Markov model can only capture the movement of a mobile node given the previous n locations (n being the Markov model's level). PCFGs however are able to take as input a set of variable length movement sequences and infer the movement patterns (with temporal information) within these sequences, no matter what the lengths of the dependence on the previous location are. Efforts in literature to deal with such shortcomings inherent with Markov models have been Sample Pattern Matching (SPM) [37], Prediction by Partial Matching (PPM) [38], and LZ-Based [22][39] approaches which were mentioned in the previous subsection as replicable with a PCFG description. Although these methods may work with variable-length history, the PCFG construction methodology given in Sec. 4 provides additional advantages. Due to multiple *chunk* operations (which find frequent movement patterns, hence already helping the understanding of the mobility properties of nodes) followed by *merge* operations, the constructed PCFG can introduce generalizations that provide further information on the mobility properties which may not even appear explicitly in the training data (an example of this is given for the parking lot simulation in Sec. 7.1.1). Furthermore, these methodologies mainly deal with the task of prediction, whereas our application of PCFGs in this paper works with full length movement sequences, hence it is a highly accurate generative model (our comparison with a Markov model based synthetic data generator is given in Sec. 7). For a predictive model that looks up to k-length (where k is variable) history, all that needs to be done is to feed movement subsequences that are seen in the training data as separate sentences, which we leave for future work.



$$
\begin{aligned}
\text{Start} &\rightarrow g_{1,1}\, V_{[f,c]}\ \text{Start}\ V_{[f,c]}\, g_{1,1}\ (p_{1,1})\ |\\
&\quad\vdots\\
&\quad g_{8,14} V_{[f,c]}\ \text{Start}\ V_{[f,c]}\, g_{8,14}\, (p_{8,14})\ |\\
&\quad g_{1,1}\ (p_{1,1})\ |\ \cdots\ |\ g_{8,14} (p_{8,14})\\
V_{[f,c]} &\rightarrow \text{Random Speed with a Preset Distribution}
\end{aligned}
$$

Fig. 6. A Palindromic Mobility Example and the PCFG that Describes It

The last advantage of PCFG-based mobility modeling that we would like to list here is its expressive capabilities due to Automata Theory. PCFGs, which are extended versions of Context Free Grammars (via assigning probability values to rules) are equivalent (in terms of describing languages) to Nondeterministic Push-down Automata [40], which utilize an unbounded stack. A Markov Model on the other hand, is equivalent to a Finite Automaton, which can only decribe Regular Languages. The variable-length MM (or other similar aforementioned methods) still build a static structure (the variable-length is due to seeing those sequences in the training data), hence they also can only describe mobility behaviors that adhere to a Regular Language. In Figure 6, we give a mobility example, which is actually very feasible in a real-world application[3]. The nodes in this example obey a *palindromic* movement constraint (i.e. they come back to the point they started through the same route). Examples of such movement include rescue operations (i.e. a fire-fighter getting into a building, and leaving it), or simply, the daily routines of most of us (usually the same path is used to go somewhere, and come back home). Such movement can only be emulated by a PCFG (or a more capable Automata, such as a Turing Machine), since the language that describes a palindrome is a non-regular one[4]. Although such non-regular languages are extremely hard to learn from training data, this example clearly demonstrates the PCFG as a superior modeling method due to its capabilities for modeling mobility properties.

3. In the figure, the terminals starting with g represent grid-locations, while the probabilities (e.g. $p_{1,1}$) are application specific, and depend both on the likelihood of visits to certain grid-locations, or on the expected length of routes.)

4. Unlike *palindrome* language, there are non-regular languages that cannot be described by a CFG, hence a PCFG as well.

7 Evaluations

In this section, we demonstrate the usefulness of our PCFG-based methodology in two mobility modeling applications: (i) extracting mobility patterns of monitored agents belonging to specific types of event data collected by a sensor network, and (ii) synthetic trace generation in mobile networks.

7.1 Modeling the Mobility Patterns of Monitored Agents in Wireless Sensor Networks

Wireless sensor networks (WSN) are often deployed to collect and process useful information about their surroundings. In this section, we propose the use of PCFG inference to model the mobility patterns embedded in sensor network measurements when these patterns are characteristic of a certain event. The main purpose of this application is to demonstrate that the PCFGs are important to recognize long-term temporal movement patterns that the previous more simplistic models are not able to infer. Furthermore, such patterns can be utilized later to recognize movements of a node and detect a potential event, which can have applications in security, health-care and domestic applications.

Sensor network applications have previously utilized PCFG methodology in literature. PCFGs are utilized to parse the actions of a user in order to infer higher level behaviors in [29]. Furthermore the authors present an assisted living application in [41]. These works do not present or utilize automated construction of grammars, rather they use manually built PCFGs. Mitomi et al. [42] provides a system to classify a temae into types by using a two-level system. In the first level, the movements of the host is detected using a camera, and then this string of movements is assigned to a temae class using a PCFG. [30] presents a visual system to recognize human gestures and to detect interactions in a parking lot environment. Finally, in [43], complex activities are recognized in a game of Blackjack. Robustness is achieved in this work by considering *insertion*, *substitution* and *deletion* errors in parsing.

7.1.1 *Real world Scenario and Simulations*

Following [1], we give an example of how PCFGs can be used in a parking lot application where events of parking at a spot can be detected. A grammar can be trained by using many examples for different possible events, taking each sentence to be the set of movements that a car performs during parking.

A simulation based on such a real world scenario is given below. The 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. We assume that one-square movement of any car lasts one time unit. Parking (that is staying in parked state) time is exponentially distributed with mean of 5000 of the same 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.

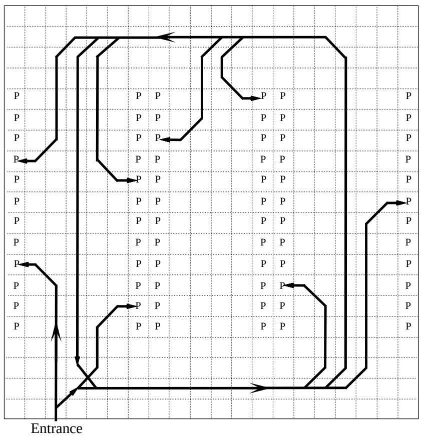


Fig. 7. Car trajectories for the event *Enter and Park*

Figure 7 shows the car trajectory associated withthe parking event. Squares with the letter P in them represent the parking spots. Due to limited space, we only present our observation of the event "Entering the parking lot, finding the closest available parking spot and then parking there (*Enter and Park*)" in this section; however, we direct the interested reader to [1] for more events that have been examined (namely, "Entering the parking lot, not being able to find an empty spot and leaving (*Enter and Leave*)", and "Leaving the parking lot from a parking spot (*Leave Parking*)"). 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% (now the movement sequence belong to the event *Enter and Leave*) and starts another circle around the parking lot with probability of 40%. 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 the event *Enter and Park* was 1468 sentences (25333 symbols).

Figure 8 gives the grammar inferred from the results of our simulations. Tokens (smallest action unit) for the grammar are: *en (enter)*, *f (one or more forward actions)*, *tr (turn right 45 degrees)*, *tl (turn left 45 degrees)* and *s (stop)*. Clearly, movement tokens are relative tokens described in Section 5.2. Furthermore, we combine multiple forward movements into a single terminal symbol (f) to simplify processing, taking advantage of the fact that a forward movement does not change direction.

```
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. 8. PCFG for event *Enter and Park*

The grammar in Figure 8 fully models the actions of a car representing parking event shown in Figure 7. In the listed grammar, the nonterminal *M1* was created by a *merge* operation, while those whose names begin with C were created by *chunk* operations. Moreover, the grammar for this event automatically discovered that the activity is recursive, i.e. there may be several circles around the parking lot before the spot is found. This is achieved by repeating *C3* in nonterminal *M1* (see Figure 8). Careful examination reveals that nonterminal *C3* reduces to $4 \cdot (f\ tl\ f\ tl)$ which completes a circle around the parking lot (therefore the grammar has recognized *a circle* as a sub-event). This illustrates the fact that a grammar can produce more than the training data contains (generalization) simply by capturing the recursiveness and is an advantage over previous mobility models (as also mentioned in Sec. 6.2), including the ones employing variable-length context. 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 the string gets smaller as it gets longer). Furthermore, unnecessary generalizations (i.e. over-generalization) that may harm the mobility capturing capabilities of the PCFG are prevented through the use of *a posteriori* probability.

7.2 Synthetic Trace Generation in Mobile Networks

In this section, we present our utilization of PCFG-based mobility modeling to generate synthetic traces. Once a PCFG is constructed from a real world trace, a large set of sentences can be produced from it, creating a synthetic mobility trace. Following the initial presentation in [2], we first discuss here how the PCFG-based mobility model can be utilized to generate node movements, leading to a realistic synthetic mobility data. Finally, we provide the evaluation of how close the generated traces are to the original trace, and the time spent in building a PCFG for different aforementioned approximation schemes. We compared our PCFG based synthetic data generation to a 2-level Markov Model based generator presented in [24]. This is not a memoryless approach and it has been shown to work better than other methods for mobility prediction. Hence, intuitively, it is also a good model for capturing properties of the actual traces. A PCFG may be seen as a Markov Model with flexible length, since it models varying length sequences. Furthermore, automated PCFG construction can achieve generalization, hence capturing more movement patterns than a Markov Model (see Sec. 6.2 for further discussions).

7.2.1 Trace Generation via the PCFG-based Mobility Model

As aforementioned, a PCFG is automatically constructed to capture spatial patterns of node movements when mobility trace consists of terminal symbols representing the locations at which a mobile node can reside. The probabilities provided in the PCFG give us the likelihood for movement patterns. Another mobility information that can be represented by a PCFG is the meeting sequences for mobile nodes. In this case however, the terminal symbols represent mobile nodes in the network. To model the temporal properties of the mobile nodes, we utilize time tokens described in Sec. 5.1. Furthermore, depending on the complexity of the mobility modeling task (size of the area, variety of the movements etc.), relative tokens (see Sec. 5.2) can also be used.

Synthetic trace generation is basically creating a random sentence from the constructed grammar, according to the production probabilities. This sentence gives both the temporal and spatial information for the single mobile node, while also encapsulating the probability of movements, via the production probabilities in the grammar. Furthermore, once the generated sequence is completed (all the nonterminals in the sentence are replaced with terminals), a new sentence can be generated for the corresponding mobile node. An important point to take into account here is that the last location (symbol) of the movement sequence (sentence) should be the same as the first location (symbol) of the new movement sequence (sentence), for continuity.

7.2.2 Evaluation of the Trace Generation Method

In this section, we are measuring similarity between real world traces and the synthetic mobility traces generated by the proposed method. We utilized two datasets: the first one [44] contains bus-to-bus meeting data collected in Amherst, MA (DieselNet - Spring

2006), while the second dataset contains the cab mobility data collected in San Francisco, CA [45]. To train the PCFG for [44], we have taken sentences to be the set of buses met during one round of a bus on the route. Each bus type in this dataset has a set route, therefore we can artificially set a start and end point (we chose those as the busiest grids in terms of the number of meetings). Hence we created the synthetic data as a set of rounds. For [45], we have divided the area into a 25x25 grid, and taken each sentence to be the set of locations (with time that passes between) until a driver gets a customer into the taxi, or the trip that is taken with the customer inside the taxi.

We used the following metrics in the comparison. For DieselNet Dataset, we have collected what buses are met by a bus on a given route right after a certain sequence of meetings. For example, the error rate *Cons k* gives the difference of a given model from the actual trace in terms of the distributions of which buses are met after a certain (k-1)-length sequence of buses are met. Hence it can be taken as the distribution difference of meeting sequences of length k. To calculate the difference, we used the Euclidean distance between the sequence distributions. In other words, given that generated data have g_i percentage of a meeting sequence i to appear, and the real world data have r_i percentage, we calculate $\Delta_{Cons} = \sqrt{\sum_{i=1}^{s}(r_i \cdot (g_i - r_i))^2}$ (where s is the number of meeting sequences of length k).

TABLE 1
Description of the Approximation Levels for Grammar Construction Utilized in Our Evaluations

	Chunk Search Appr. Level	Merge Search Appr. Level
Appr. Level 0	0	0
Appr. Level 1	1	0
Appr. Level 2	0	1
Appr. Level 3	1	1
Appr. Level 4	0	2
Appr. Level 5	1	2

Another metric is based on the inter-meeting times, in which we calculate the time it takes for a bus to meet another bus given it has met a certain sequence of buses. *Intern k* denotes the time it takes for a bus to meet a k^{th} bus after meeting k-1 buses in a sequence. We use the weighted Euclidean distance between the average intermeeting times to calculate errors. In other words, given that generated data have an average intermeeting time $tg_{i,s}$ for bus b_i and the real world data have an average intermeeting time $tr_{i,s}$ for bus b_i after meeting a certain bus in the k-length sequence s, we calculate $\Delta_{Intern} = \sqrt{\sum_{i=1}^{p}\sum_{s=1}^{r}(w_{i,s} \cdot (tg_{i,s} - tr_{i,s}))^2}$ (where p is the number of buses, r is the number of meeting sequences that end with bus b_i, and $w_{i,s}$ is the weight of the sequence s ending with bus b_i, calculated according to the frequency of meeting sequences). For the taxi mobility dataset, we use the same metrics, however the buses are replaced with the location grids, hence *Cons 3* for the location distributions means the error on the distribution of three sequences of locations that a mobile node goes through.

In this section, aside from comparing the PCFG-based mobility modeling to a Level-2 MM, we also compare the approximation schemes that were introduced in Section 4. Table 1 provides a description of what we mean by *Appr. Level k*. In the table, *level-0* in *Merge Search* means that all pairs of nonterminals and their effect on all the rest of the grammar are evaluated for merge operand selection. *level-1* means again the consideration of all nonterminal pairs, however only their effect on each other is evaluated. *level-2* in *Merge Search* applies our highest level of approximation to finding merge nonterminal pairs, as presented in Section 4.7, which chooses the two nonterminals with smallest total rule frequencies. In *Chunk Search*, the approximation *level-0* means the search for the most advantageous chunk operator, and considering all lengths. Again, *level-1* represents our highest level of approximation to find the chunk string, as described in Section 4.4, which looks for strings of length up to 5.

The results for the DieselNet Dataset [44] are presented in Table 2. For the synthetic trace generation purposes, we utilized the best grammar that was obtained during the grammar construction process. This means that during the construction, we also back up the grammar with the highest *a posteriori* so far. *Best Grammar Construction Time* metric in the table represents the last time when this back up is done (i.e. after this second, the grammar's *a posteriori* has gone below, but the grammar construction process continues, as in Algorithm 1). Furthermore, we limited the number of consequent *merge* operations that can be done (between 100 and 500), for reasons of efficiency, since after a certain number of merges on the grammar, there are no advantages that are gained in terms of grammar goodness. As it can be observed from Table 2, although the grammars take longer[5] to construct and generate synthetic data, they provide traces that are much closer (up to 93%) to the original trace (as shown by *Intern* and *Cons* metrics) than the Markov model. The times in the table are consistent with the $O(D^2 \log D)$ time complexity for PCFGs, for Appr. Level 5 ($O(D^4)$ for Level 0). The complexity for building a Markov Model is only $O(D)$, since it requires a single pass over the training data to calculate transition probabilities. Furthermore, as expected, our approximations significantly cut grammar construction times. An improvement in the goodness of the data generated is also observed, which is counter-intuitive. This can be explained by the fact that low-approximation schemes actually find *merge*

5. For our experiments, we implemented the algorithms in Perl, and utilized a PC with 2.8GHz Intel i7 processor and 8GB of RAM, running Ubuntu.

TABLE 2
Evaluation Results for DieselNet Dataset [44]

	Appr. Level 0	Appr. Level 1	Appr. Level 2	Appr. Level 3	Appr. Level 4	Appr. Level 5	Markov Model Level-2
Construction Process Length (sec)	147453	11973	2864	3936	1894	154	0.108844
Best Grammar Construction Time (sec)	9838	397	1604	76.62	1541	79.37	NA
Synthetic Trace Generation Time (sec)	94.98	99.12	107.07	104.20	100.44	108.81	31.59
Cons/Intern 2	0.002/1.8	0.002/1.45	0.002/1.47	0.002/1.26	0.002/1.26	0.002/1.28	0.018/11.46
Cons/Intern 3	0.003/3.06	0.002/2.56	0.002/1.6	0.002/1.95	0.002/1.88	0.002/1.76	0.016/9.40
Cons/Intern 4	0.004/4.74	0.003/3.23	0.003/1.73	0.002/1.89	0.003/1.87	0.003/2	0.03/40.53
Cons/Intern 5	0.005/5.2	0.004/3.84	0.003/1.91	0.003/1.79	0.003/1.98	0.003/2	0.041/52.62
Cons/Intern 6	0.004/4.88	0.003/3.72	0.003/1.82	0.003/1.67	0.003/1.85	0.003/1.68	0.041/60.03

TABLE 3
Evaluation Results for the First 500 Routes in Taxi Mobility Dataset [45]

	Appr. Level 0	Appr. Level 1	Appr. Level 2	Appr. Level 3	Appr. Level 4	Appr. Level 5	Markov Model Level-2
Construction Process Length (sec)	26947	7860	281.48	97.93	60.89	20.04	0.014
Best Grammar Construction Time (sec)	9690	1685	63.73	6.53	48.67	8.34	NA
Synthetic Trace Generation Time (sec)	3.52	3.85	3.87	5.44	3.71	4.79	0.92
Cons/Intern 2	0.008/46.45	0.019/64.12	0.005/8.81	0.005/8.92	0.005/12.37	0.004/10.39	0.099/58.45
Cons/Intern 3	0.015/82.76	0.033/123.07	0.006/14.95	0.007/11.98	0.007/16.98	0.007/16.55	0.043/74.74
Cons/Intern 4	0.017/121.61	0.024/136.58	0.007/14.5	0.008/13.01	0.008/16.25	0.008/14.51	0.058/220.83
Cons/Intern 5	0.018/151.48	0.021/150.01	0.008/12.04	0.01/10.53	0.009/12.15	0.009/14.14	0.071/269.92
Cons/Intern 6	0.02/128.6	0.022/92.24	0.009/12.17	0.011/10.89	0.011/11.24	0.011/14.54	0.08/290.57

operations that are good for grammar *a posteriori*, however not beneficial for data generation, since it generalizes the grammar. This generalization brings movement sequences which are not seen in training data, which is useful in certain cases (as in our parking lot example), but not so much in others.

For the Taxi Mobility Dataset [45], we utilized both the whole dataset and the first 500 routes in it. The processed dataset consists of 460000 routes (a route is a set of locations and movements, each separated by a length of time), and it was impossible for us to evaluate all approximation schemes (due to construction time) on such a large set. However, the ability to deal with large datasets comes with the approximations, hence we compare the *Appr. Level-5* of grammar construction to the Markov model using the whole dataset. However, we utilize the first 500 routes to compare all 6 approximation schemes (*Appr. Level 0-5*) and the Markov model. The results for the first 500 routes in Taxi Mobility Dataset [45] are shown in Table 3. Again, it can be observed that our approximations provide a much much faster way of grammar construction, while improving in terms of synthetic data closeness (demonstrated by *Intern* and *Cons* metrics). The reasons for this phenomena are similar to the ones given for the UMASS Bus Data results. Furthermore, as presented both for the first 500 routes (Table 3), and for the whole dataset (Table 4), the PCFG-based mobility modeling provide mobility generation much closer (up to 95%) to the actual trace than the Markov model, although it takes longer to construct the model and generate traces.

TABLE 4
Evaluation Results for the Whole Taxi Mobility Dataset [45]

	Appr. Level 5	Markov Model Level-2
Construction Process Length (sec)	361983	11.74
Best Grammar Construction Time (sec)	331092	NA
Synthetic Trace Generation Time (sec)	1179	1.58
Cons/Intern 2	0.004/41.68	0.076/67.08
Cons/Intern 3	0.006/69.51	0.035/57.78
Cons/Intern 4	0.007/103.92	0.059/125.91
Cons/Intern 5	0.008/137.8	0.078/166.70
Cons/Intern 6	0.01/159.25	0.091/212.57

8 Conclusion

In this paper, we have focused on the usefulness of PCFG framework for modeling mobility properties of nodes in data collected by or generated by a network. While previous publications also applied the scheme to various domains, our approach differs from them because we are using training data to derive a precise grammar automatically. We presented a fast probabilistic context free grammar inference method for modeling mobility of nodes. We have also discussed two application domains to demonstrate the usefulness of our mobility modeling approach: (i) extraction of mobility patterns belonging to event data in sensor networks, and (ii) synthetic trace generation in mobile networks. We provided evaluations based on simulations and real world traces for these two domains. The results allow us to conclude that PCFG modeling is a compact and efficient way of representing network mobility data without losing their properties. The created compact representation can be used later to generate the data with the same properties as the original one, as well as to provide predictions for and insights into the applications from which the mobility patterns originated.

References

[1] Geyik, S. C., Szymanski, B., *Event Recognition in Sensor Networks by Means of Grammatical Inference*, IEEE INFOCOM, 2009.

[2] Geyik, S. C., Bulut, E., Szymanski, B. K., *PCFG Based Synthetic Mobility Trace Generation*, IEEE Globecom, 2010.

[3] Broch, J., Maltz, D. A., Johnson, D. B., Hu, Y. -C., Jetcheva, J., *A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols*, ACM MobiCom, 1998.

[4] *Universal Mobile Telecommunications System (UMTS); Selection Procedures for the Choice of Radio Transmission Technologies of the UMTS (UMTS 30.03 version 3.2.0)*, ETSI TR 101 112 V3.2.0 (1998).

[5] Ahmed, S., Karmakar, G. C., Kamruzzaman, J., *An Environment-Aware Mobility model for Wireless Ad Hoc Network*, Elsevier Computer Networks, Vol. 54, No. 9, 2010.

[6] Harfouche, L., Boumerdassi, S., Renault, E., *Towards a Social Mobility Model*, IEEE PIMRC, 2009.

[7] Hu, L., Dittmann, L., *Heterogeneous Community-Based Mobility Model for Human Opportunistic Network*, IEEE WiMob, 2009.

[8] Wang, J., Yuan, J., Shan, X., Feng, Z., Geng, J., You, I., *SaMob: A Social Attributes Based Mobility Model for Ad Hoc Networks*, IEEE IMIS, 2011.

[9] Calegari, R., Musolesi, M., Raimondi, F., Mascolo, C., *CTG: A Connectivity Trace Generator for Testing the Performance of Opportunistic Mobile Systems*, ESEC/FSE, 2007.

[10] Frangiadakis, N., Kyriakakos, M., Hadjiefthymiades, S., Merakos, L., *Realistic Mobility Pattern Generator: Design and Application in Path Prediction Algorithm Evaluation*, IEEE PIMRC, 2002.

[11] Karnadi, F. K., Mo, Z. H., Lan K., *Rapid Generation of Realistic Mobility Models for VANET*, IEEE WCNC, 2007.

[12] Tan, D. S., Zhou, S., Ho, J., Mehta, J., Tanabe, H., *Design and Evaluation of an Individually Simulated Mobility Model in Wireless Ad Hoc Networks*, CNDS, 2002.

[13] Chang, Y., Liao, H., *EMM: an Event-Driven Mobility Model for Generating Movements of Large Numbers of Mobile Nodes*, Simulation Modelling Practice and Theory, Vol. 13, Iss. 4, 2005.

[14] Kim, M., Kotz, D., Kim, S., *Extracting a Mobility Model from Real User Traces*, IEEE INFOCOM, 2006.

[15] Tuduce, C., Gross, T., *A Mobility Model based on WLAN Traces and its Validation*, IEEE INFOCOM, 2005.

[16] Hsu, W., Spyropoulos, T., Psounis, K., Helmy, A., *Modeling Time-Variant User Mobility in Wireless Mobile Networks*, IEEE INFOCOM, 2007.

[17] Maeda, K., Uchiyama, A., Umedu, T., Yamaguchi, H., Yasumoto, K., Higashino, T., *Urban Pedestrian Mobility for Mobile Wireless Network Simulation*, Ad Hoc Networks, Vol. 7, Iss. 1, 2009.

[18] Gunasekaran, S., Nagarajan, N., *An Improved Realistic Group Mobility Model for MANET Based On Unified Relationship Matrix*, IEEE IACC, 2009.

[19] Koberstein, J., Peters, H., Luttenberger, N., *Graph-Based Mobility Model for Urban Areas Fueled with Real World Datasets*, Simutools, 2008.

[20] Ashbrook, D., Starner, T., *Using GPS to Learn Significant Locations and Predict Movement Across Multiple Users*, Journal of Personal and Ubiquitous Computing, Vol. 7, Iss. 5, 2003.

[21] Chinchilla, F., Lindsey, M., Papadopouli, M., *Analysis of Wireless Information Locality and Association Patterns in a Campus*, IEEE INFOCOM, 2004.

[22] Bhattacharya, A., Das, S. K., *LeZi-Update: An Information-Theoretic Approach to Track Mobile Users in PCS Networks*, ACM/Kluwer Wireless Networks, Vol. 8, Iss. 2-3, 2002.

[23] Song, L., Kotz, D., *Evaluating Location Predictors with Extensive Wi-Fi Mobility Data*, IEEE INFOCOM, 2004.

[24] Song, L., Deshpande, U., Kozat, U. C., Kotz, D., Jain, R., *Predictability of WLAN Mobility and Its Effects on Bandwidth Provisioning*, IEEE INFOCOM, 2006.

[25] Lelescu, D., Kozat, U. C., Jain, R., Balakrishnan, M., *Model T++: An Empirical Joint Space-Time Registration Model*, ACM MobiHoc, 2006.

[26] Jurafsky, D., et al., *Using a Stochastic Context-Free Grammar as a Language Model for Speech Recognition*, ICASSP, 1995.

[27] Manning, C. D., Schutze, H., *Foundations of Statistical Natural Language Processing*, Cambridge, Mass.: MIT Press, 1999.

[28] Sakakibara, Y., *Grammatical Inference in Bioinformatics*, IEEE Trans. on Patt. Analysis and Mach. Intelligence, Vol. 27, Iss. 7, 2005.

[29] Lymberopoulos, D., Ogale, A. S., Savvides, A., Aloimonos, Y., *A Sensory Grammar for Inferring Behaviors in Sensor Networks*, IPSN, 2006.

[30] Ivanov, Y. A., Bobick, A. F., *Recognition of Visual Activities and Interactions by Stochastic Parsing*, IEEE Trans. on Patt. Analysis and Machine Intelligence, Vol. 22, Iss. 8, 2000.

[31] Lari, K., Young, S. J., *The Estimation of Stochastic Context-Free Grammars using the Inside-Outside Algorithm*, Computer Speech and Language, Vol. 4, 1990.

[32] Baker, J., *Trainable Grammars for Speech Recognition*, Journal of the Acoustical Soc. of America, Vol. 65, Iss. S1, 1979.

[33] Stolcke, A., *Bayesian Learning of Probabilistic Language Models*, Doctoral dissertation, Dept. of EECS, UC at Berkeley, 1994.

[34] Chen, S. F., *Building Probabilistic Models for Natural Language*, Doctoral dissertation, Dept. of CS, Harvard University, 1996.

[35] Laplace, P., *Theorie Analytique des Probabilites*, Courcier, Paris, 1812.

[36] Brodal, G., Lyngso, R. , Ostlin, A., Pedersen, C., *Solving the String Statistics Problem in Time O(n log n)*, ICALP, 2002.

[37] Jacquet, P., Szpankowski, W., Apostol, I., *A Universal Predictor Based on Pattern Matching*, IEEE Trans. on Information Theory, Vol. 48, Iss. 6, 2002.

[38] Cleary, J., Teahan, W., *Unbounded Length Contexts for PPM*, Computer Journal, Vol. 40, 1997.

[39] Gopalratnam, K., Cook, D. J., *Online Sequential Prediction via Incremental Parsing: The Active LeZi Algorithm*, IEEE Intelligent Systems, Vol. 22, Iss. 1, 2007.

[40] Sipser, M., *Introduction to the Theory of Computation*, PWS Publishing, Second Ed., 2006.

[41] Teixeira, T., Lymberopoulos, D., Culurciello, E., Aloimonos, Y., Savvides, A., *A Lightweight Camera Sensor Network Operating on Symbolic Information*, Workshop on Distributed Smart Cameras, ACM SenSys, 2006.

[42] Mitomi, H., Fujiwara, F., Yamamoto, M., Taisuke, S., *Bayesian Classification of a Human Custom Based on Stochastic Context-Free Grammar*, Sys. and Comp. in Japan, Vol. 38, Iss. 9, 2007.

[43] Moore, D., Essa, I., *Recognizing Multitasked Activities from Video using Stochastic Context-Free Grammar*, AAAI-02, 2002.

[44] Zhang, X., Kurose, J., Levine, B., N., Towsley, D., Zhang, H., *Study of a Bus-based Disruption Tolerant Network: Mobility Modeling and Impact on Routing*, ACM Mobicom, 2007.

[45] Piorkowski, M., Sarafijanovoc-Djukic, N., Grossglauser, M., *A Parsimonious Model of Mobile Partitioned Networks with Clustering*, COMSNETS, 2009.

Sahin Cem Geyik is currently a Ph.D Candidate in the Computer Science Department, at Rensselaer Polytechnic Institute (RPI), Troy, NY. He received his BS degree in Computer Engineering from Bogazici University in Istanbul, Turkey, in 2007.
Eyuphan Bulut received his Ph.D degree in Computer Science from Rensselaer Polytechnic Institute (RPI), Troy, NY, in 2011.
Boleslaw K. Szymanski (M'82-F'99) is the Claire and Roland Schmitt Distinguished Professor of Computer Science at Rensselaer Polytechnic Institute (RPI), Troy, NY. He received his Ph.D. degree in computer science from the National Academy of Sciences, Warsaw, Poland, in 1976.