NETWORK DATA MODELING VIA GRAMMATICAL STRUCTURES

By

Sahin Cem Geyik

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: COMPUTER SCIENCE

Approved by the
Examining Committee:

Professor Boleslaw K. Szymanski, Thesis Adviser

Professor Christopher D. Carothers, Member

Assistant Professor Sanmay Das, Member

Associate Professor Koushik Kar, Member

Professor Alex Pentland, Member

Rensselaer Polytechnic Institute
Troy, New York

June 2012
(For Graduation August 2012)

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Data modeling in computer networks means finding a compact and faithful representation of the data on which network applications work. This data can range from the information that is being transferred in a wireless network to mobility information of nodes in a mobile network, or to the observations of the entity behaviors recorded by the network, etc. In any kind of application, data modeling holds an important place if high efficiency is a requirement.

In this thesis, we examine the use of grammatical structures, with a high emphasis on probabilistic context free grammars (PCFG), as the modeling framework for such data. Informally, PCFGs are regular context free grammars where production rules are assigned a probability value, representing how likely these rules are to be used when generating a sentence from this grammar. Utilization of PCFGs includes initially deriving the grammatical structure from the real world traces, and later applying the structure for the necessary purposes of the application. These purposes include saving disk space in space-constrained systems (such as sensor networks), classification and recognition of observed events, facilitating manual inspection (i.e. improving interpretability), etc. The subject matter presented in this thesis contains both the grammar construction process as well as different domains to which the grammatical structure can be applied.

The contributions of this PhD thesis can be summarized as follows:

- An automated Probabilistic Context Free Grammar (PCFG) construction algorithm, which lowers the time complexity of the previous approaches. We also evaluate (both theoretically and empirically) how much of an advantage we provide over the previous approaches complexity-wise, and the comparison of the goodness of grammars constructed with multiple methods.

- Application of the PCFG learning and later processing approach to different domains of Network Data Modeling. These domains include:

  - Event Recognition in Sensor Networks,
  - Mobility Modeling and Synthetic Trace Generation for Mobile Networks,
  - Behavior Modeling in Social Networks, and,
  - Learning of Service Composition Rules for Service Oriented Architecture in Sensor Networks.

- We also contribute to metric-based service composition in sensor networks, as well as to application of switch options in pervasive sensor applications.

CHAPTER 1
Introduction

Data Modeling in Computer Networks (not to be confused with the same term in Software Engineering) means finding a compact and faithful representation of the data on which network applications work. This task includes finding a model of the given data that includes enough features to construct samples synthetically with the same statistical properties as the input, as well as to decide whether a future occurrance belongs to the same class as the previously monitored data. The motivation of such modeling can be presented in the following advantages:

- Saving disk-space in space-constrained systems such as wireless sensor networks etc.,
- Processing of the models gives us recognition benefits; i.e. we can detect if a certain sequence of data is a member of the model or not,
- Decreasing communication cost since we can compress the data that needs to be transmitted,
- Facilitating manual inspection, i.e. improving interpretability, since data modeling can capture a subset of an event which is significant for a certain purpose.

The focus of this thesis is on data modeling in networking applications by utilizing grammatical structures, with a special emphasis on *Probabilistic Context Free Grammars* (*PCFG*). The introduction of this thesis will firstly provide several modeling methodologies with their applications in the literature. Later, we will introduce Probabilistic Context Free Grammars, with a short literature listing on their applications.

* Portions of this chapter previously appeared as: S. C. Geyik and B. K. Szymanski, "Event recognition in sensor networks by means of grammatical inference," in *Proc. IEEE INFOCOM*, 2009, pp. 900−908.

1.1 Network Data Modeling Techniques in Literature

In this section we will list the following data modeling techniques (with a short definition and example applications in the literature): (*i*) Frequent Pattern Mining, (*ii*) Markov Models, and (*iii*) Latent Dirichlet Allocation.

1.1.1 Frequent Pattern Mining

Frequent Pattern Mining [1]-[3] examines the co-occurrence of items in large databases. The discovery of items occurring together means that this may have a semantic meaning (e.g. people who purchase a certain item may buy another item at the same time more frequently) in certain applications. Such co-occurrence can be ordered (sequence mining) or independent of the order (itemset mining). A few common algorithms for the discovery of frequent patterns can be listed as: the *Apriori* algorithm [4] (itemset), the *ECLAT* algorithm [5] (itemset), the *FPGrowth* algorithm [6] (itemset), *GSP* algorithm (sequence, similar to *Apriori*), SPADE [7] (sequence), *projection-based* algorithm [8] (sequence) and *Ukkonens Linear Time Algorithm* for suffix trees [9] (sequence).

Frequent pattern mining is applied in several domains such as marketing analysis [1], web mining [10], [11], computational biology [12], security [13] etc.

1.1.2 Markov Models

Markov Models are probabilistic state transition models which obey the Markov rule: the next state of the system is only dependent on the previous k-states of the system. We are focusing on two types of Markov models in this section: (*i*) Markov Chains (MC), and (*ii*) Hidden Markov Models (HMM). Basically, Markov Chains are systems where the actual state is observable and known, while Hidden Markov Models are the systems in which only the output is seen, but the actual state is not. Hence, in HMMs, prediction of the current state is a significant problem, and most of these systems assume the Markov property (i.e. the system follows a Markov process), since the actual states cannot be seen.

While the construction of an MC is pretty straight-forward via statistical measurements, more sophisticated algorithms are required in order to process and construct HMMs. A very well known algorithm to learn the parameters of an HMM is *Baum-Welch* Algorithm [14], which is a special type of the *Expectation Maximization* (EM) Algorithm. The parameters to be learned are basically the emission parameters which determine the probability of a certain output given the state, and the transition parameters that provide the probability that the system changes its state from one given state to the other. Furthermore, the *Viterbi* algorithm [15] gives the most likely set of hidden states given a set of outputs, while the *Forward-Backward* Algorithm provides the probability of a certain hidden state given a sequence of outputs.

Markov Models have been applied to many different domains such as language modeling [16], [17], speech recognition [18], [19], visual recognition [20], mobility modeling [21], [22], computational biology [23], [24], etc.

1.1.3 Latent Dirichlet Allocation

Latent Dirichlet Allocation (LDA) [25] is a generative model which models the data from a specific application domain into a set of documents. Each document consists of a set of words, created by a topic. Each topic follows a probability distribution on the set of words that it can generate, and each document has a probability distribution on the set of topics, which generate the words that it includes. According to this model, a document is generated as follows [25]:

- Choose the number of words for this document according to a Poisson distribution.

- Choose a Multinomial(θ) topic distribution for this document. Multinomial(θ) is chosen according to a Dirichlet distribution with the α parameter (Dir(α)).

- For each word (the order is not important for LDA, hence it is based on the "bag-of-words" assumption), first choose a topic (according to Multinomial(θ) chosen for this document) that this word is created from. Then, generate the word from the chosen topic according to the Multinomial distribution of words (specific to each distinct topic, i.e. every topic has a different probability distribution on the set of words that it can generate) that can be created by this topic.

As shown in the original paper [25], it is intractable to calculate the parameters of the model from training data. Approximate methods such as a simple convexity-based variational approach [25], a Markov chain Monte Carlo algorithm [26], Gibbs sampling [27], and expectation propagation [28] have been utilized in the literature.

Latent Dirichlet Allocation and its extensions (e.g. Spatial-LTM [29], multi-level topic model [30], semi-latent topic model [31]) have been used in various application domains. To name a few, text mining [25], visual recognition and processing [29], [31], security [32], [33], social networks [30], [34], [35] (e.g. social role discovery, classification of human routines and interaction), software analysis [36], etc.

1.2 Probabilistic Context Free Grammars

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

- $\mathrm{S_t}$ is a list of terminal symbols, which form the sentences produced by this grammar,
- $\mathrm{S_{nt}}$ is a list of nonterminal symbols, which are replaced by rules in R during sentence production,
- Start is the initial nonterminal symbol for any sentence produced by the grammar,
- 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 which belong to the same nonterminal.

A PCFG is simply a context free grammar with a probability assignment to the production rules. Production probability of a sentence generated by a PCFG is calculated by multiplying the probability values on each branch of its parsing tree. See the following example for a simple PCFG which produces strings of type $a\ b^n$ or $b\ c^n$.

$$\begin{aligned} Start &\rightarrow a\ N\ (0.4)\ \mid b\ M\ (0.6) \\ N &\rightarrow b\ (0.3)\ \mid b\ N\ (0.7) \\ M &\rightarrow c\ (0.4)\ \mid c\ M\ (0.6) \end{aligned}$$

Figure 1.1 shows the parse tree of the string $b\ c\ c$ for the above grammar. As it can be seen, this string has the probability of 0.6 x 0.6 x 0.4 $=$ 0.144.

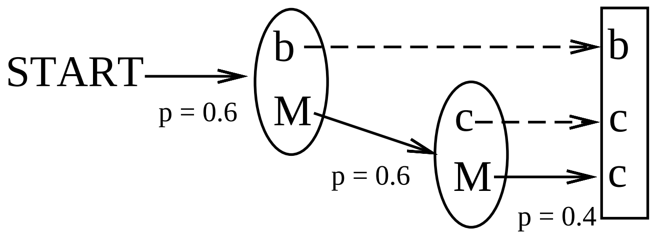


Figure 1.1: Parse Tree for the String b c c

Next, we will give a short literature survey on the applications of PCFGs in different domains.

1.2.1 Applications of PCFGs in Literature

Probabilistic Context Free Grammars have many uses in speech recognition [37], [38], natural language processing [16], computational biology [39], [40] etc.

A major area in network application of PCFGs is the recognition of events in sensor networks. In [41], the authors use a PCFG to parse the actions of a user in order to infer higher level behaviors. Furthermore, they take into account the different combinations of actions for a behavior, hence set the grammars accordingly. Following the work in [42], which introduces Address-Event Imagers (limited capacity image sensors), the authors present an assisted living application in [43].

This paper describes the recognition of domestic activities from the positions of people and utilizes PCFGs for this purpose. These works do not present or utilize automated construction of grammars, rather they use manually built PCFGs. Finally in [44], a human behavior parsing sensor system is described from start to end. Experimental results are presented together with the grammars themselves, which are defined in a varying hierarchy of event recognition steps.

Mitomi et al. [45] 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. [46] presents a visual system to recognize human gestures and to detect interactions in a parking lot environment. Authors utilize an Earley-Stolcke parser [17], [47] along with a probability threshold as well as a sliding window to limit the number of branches in the parsing process. In [48], complex activities are recognized in a game of Blackjack. Again, Earley-Stolcke parsing is applied to sequences, and robustness is achieved by considering only three types of errors: (*i*) *insertion* of a token into the sequence, (*ii*) *substitution* of a token from a sequence with another one, and (*iii*) *deletion* of a token from a sequence. Finally, in [49], the authors transform hierarchical PCFG into a hierarchical Bayesian Network and use deleted interpolation (DI) [16] 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.

A last application we would like to list for the use of PCFGs is fault detection in computer systems. Paper in [50] presents a system called *Pinpoint* which provides detection of faults in the application layer of internet services. This detection is realized by constant monitoring of software component interactions and query component paths (set of components called for a query). The training stage of the system provides Pinpoint with steady-state behavior of the system while in the run-time of Pinpoint, faults are recognized due to behavior that is too divergent from the general case.

1.3 Rest of the Thesis

Following the introduction provided in this chapter, the remainder of the thesis is organized as follows. The next chapter (Chapter 2) presents our efforts in the automated construction of PCFGs, which lists our improvement in terms of time complexity and provides an evaluation of the goodness of the grammars constructed by our proposed scheme. Chapter 3 gives the application of PCFGs (and the advantages of automated construction) in the domain of event recognition in sensor networks. We propose a mobility modeling approach utilizing PCFGs in Chapter 4. Social networks behavior modeling, and service composition learning in service oriented sensor networks, both via PCFGs, are presented in Chapters 5 and 6, respectively. We also discuss our efforts in metric-based service composition in sensor networks, as well as our work in application of switch options in pervasive sensor applications in Chapter 6. Finally, conclusions of the thesis and future work are discussed in Chapter 7.

CHAPTER 2
Inference of Probabilistic Context Free Grammars (PCFG)

Probabilistic Context Free Grammars (PCFG), the introduction to which is given in Chapter 1, constitutes the main focus of this thesis. We reserve this chapter to automated construction of PCFGs from given training data. We will begin with a short related work section which describes previous efforts in PCFG inference. We will later present the algorithm which was introduced by us in [51]. We next provide our extensions to the standard PCFG definition which are required for the application of PCFGs to different network data modeling domains. We will complete the chapter by providing the improvement of our proposed scheme over previous grammar inference algorithms, and provide experiments on how our proposed PCFG inference improvements perform in terms of the goodness of the grammars constructed, and the time that is spent during this construction process.

2.1 Construction of PCFGs in Literature

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 [52], [53] serves as the solution.

For the second case, two previous works, which also form the basis for our algorithm, are relevant. The first one [17], [54] 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. This work [17], [54] also focuses on the inference of *HMMs* and *n-gram* models

The second work [55] begins with a very general grammar (as opposed to [17]

* Portions of this chapter previously appeared as: S. C. Geyik and B. K. Szymanski, "Event recognition in sensor networks by means of grammatical inference," in *Proc. IEEE INFOCOM*, 2009, pp. 900−908.

which generalizes and miniaturizes the initial specific grammar with the *merge* and *chunk* operators). Later, this grammar is made specific to the training data with five operations (*concatenation*, *classing*, *repetition*, *smoothing* and *specialization*). The authors utilize a Bayesian evaluation of the grammar, which we also used in our algorithm. This evaluation replaced the complex evaluation method in [17] (based on the symmetrical Dirichlet distribution over the PCFG model parameters) in our proposed inference algorithm.

2.2 PCFG Inference Algorithm

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.

2.2.1 Operations Used for Grammar Construction

This section explains the method introduced initially by Stolcke [17]. Grammar construction consists of two steps: sample incorporation, and application of operators.

2.2.1.1 Sample Incorporation

Sample incorporation is the initial step of constructing the grammar from training data. The training data is a set of sentences which are assumed to have been produced by the grammar we are trying to construct. Each sentence is a string of terminal symbols. These terminal symbols are reduced by the grammar to nonterminals of the form:

$$N_i \rightarrow symbol_i \text{ (frequency of this symbol)} ,$$

where *frequency* of a rule is the number of times the terminal symbol appears in the training data. The goal is to separate *terminal* productions from *nonterminal* productions. The sentences are reduced in the initial grammar by rules of the form:

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

For operational simplicity, the production frequencies are held, rather than the probabilities. Later, the *maximum likelihood* estimate of any rule's probability can be calculated as:

$$P(rule_i) = \frac{frequency(rule_i)}{\sum\limits_{k=1}^{n} frequency(rule_k)} \ , \tag{2.1}$$

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

2.2.1.2 Operators

Two operators: merge and chunk [17], are used to build the grammar step by step. We explain how these two operators work by examples.

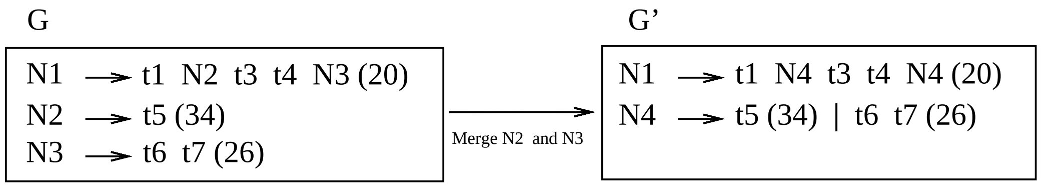


Figure 2.1: An Example of Merge Operation

Merge takes two nonterminals and reduces them into a new nonterminal by combining their rules, as well as the frequencies. See Figure 2.1 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* $\rightarrow \ldots$ *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 a 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* $\rightarrow$ *X2* (while merging *X1* and *X2*), then the new nonterminal (*X3*) would contain the rule *X3* $\rightarrow$ *X3*, which is removed together with its frequency.

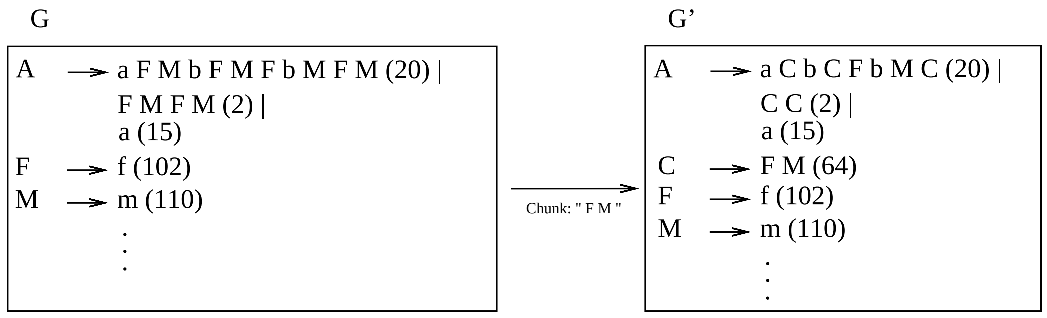


Figure 2.2: An Example of Chunk Operation

Chunk operator creates a new nonterminal with a single rule which is a string composed of symbols in 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 2.2, 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.

2.2.2 Evaluation Metric for the PCFG

It is a crucial task to evaluate the *goodness* of the grammar that is constructed from the training data $D = \{d_1, d_2, \dots\}$. For this purpose we utilize the *a posteriori* probability ($P(G|D)$) as given in [17], [55]:

$$P(G|D) = \frac{P(G)P(D|G)}{P(D)} \ , \tag{2.2}$$

and we can remove the prior for the training data from the above formulation. Hence, the evaluation metric becomes $P(G) \times P(D|G)$. Here, $P(G)$ is the grammar *a priori* probability which is inversely related to the grammar description length, denoted as l_G. Hence, based on *Occam's Razor* principle, the shorter the description length, the higher the probability. 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):

$$P(G) = \alpha \ . \ 2^{-l_G}. \tag{2.3}$$

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

$$P(D|G) = \prod_{i=1}^{|D|} p(d_i|G). \tag{2.4}$$

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 $l_s + ((number\ of\ symbols\ in\ the\ rule) \times l_s)$, 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.

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

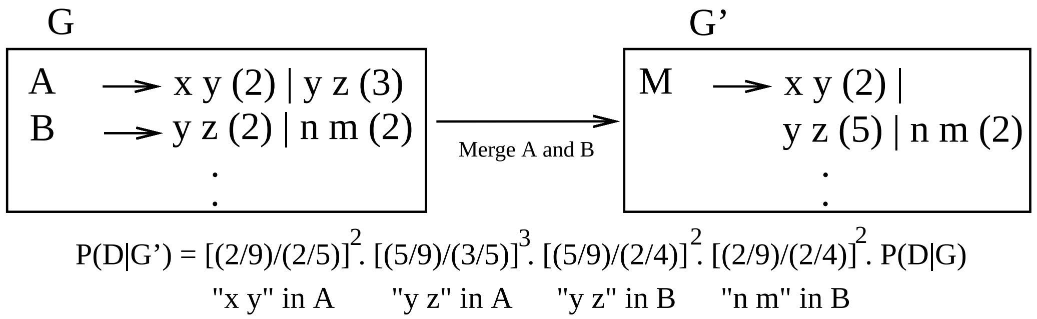


Figure 2.3: Calculation of P(D|G) for Merge Operation

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 2.2.5.

2.2.3 Computation of the Evaluation Metric

In this section, we describe the methods that we use to calculate the effect of the operators (Section 2.2.1.2) on the *a posteriori* probability (Eq. 2.2).

2.2.3.1 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 2.2. 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.

2.2.3.2 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 2.3, 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).

2.2.4 Search for the Best Merge and Chunk Arguments

As discussed in Section 2.2.3.2, 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 2.2.7.

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:

$$l_G - l_{G'} = l_s[(n-1)(k-1) - 3]. \tag{2.5}$$

2.2.5 Inference Algorithm

In most cases, merge operation decreases *a posteriori* probability according to definition of $P(G|D)$ given by Eq. 2.2. 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 keep 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. Please note that different strategies can be applied, which may allow for further generalization, or keep *a posteriori* probability always at higher values.

We will next 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.

Theorem 1. *A chunk operation neither eliminates feasible merge operations nor changes their impact on the* a posteriori *probability of the grammar.*

Proof. A chunk operation increases the number of nonterminals by introducing a new nonterminal which consists of a single rule: a string of nonterminals with probability 1.0. 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 $P(D|G)$ 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 1.0. Therefore, it does not affect $P(D|G)$, 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 operations 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. This proof furthermore justifies our *chunk* merge* choice, which is a distinct diversion from [17], where the step can be seen as *chunk merge**.

2.2.6 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 $O(l_G \times$

$l_{max} \times \log(l_G))$, 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 [17] 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 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 $T_W(D) = \sum_{i=1}^{D} O(D^3) = O(D^4)$, where D denotes the total length of the training data measured in number of symbols.

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) [56]. 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.

2.2.7 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 [17]. 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 side by side, making the change in the *a posteriori* probability as follows:

$$P(G|D)_{new} = 2^{l_s} \times P(G|D)_{prev} \times$$

$$\left(\frac{x_1/[S_X + S_Y]}{x_1/S_X} \right)^{x_1} \times \ \dots \ \times \left(\frac{x_n/[S_X + S_Y]}{x_n/S_X} \right)^{x_n} \times$$

$$\left(\frac{y_1/[S_X+S_Y]}{y_1/S_Y}\right)^{y_1} \times \ ... \ \times \left(\frac{y_m/[S_X+S_Y]}{y_m/S_Y}\right)^{y_m}$$

$$= 2^{l_s} \times P(G|D)_{prev} \times \left(\frac{S_X}{S_X+S_Y}\right)^{S_X} \times \left(\frac{S_Y}{S_X+S_Y}\right)^{S_Y} \quad .$$

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

Theorem 2. *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. $\square$

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

2.3 Extensions to the PCFG

In this section we extend the simple PCFG definition to apply it to certain domains. First, we introduce time tokens, which represent the temporal properties of the application domain. The second one is the addition of relative tokens. Relative tokens basically change the state of a modeled object according to its last state. They reduce complexity related to high number of terminals.

2.3.1 Time Tokens

To represent the temporal properties of any application domain, we utilize a special time terminal symbol, t. We are proposing two versions of this time symbol: (i) A constant time representing time symbol, where each t in the grammar represents the same amount of time, and different time amounts can be implemented with a varying number of consecutive t's, and (ii) A time distribution representing time symbol, where each symbol t is accompanied by the mean and standard deviation of the distribution of the time period that it represents.

2.3.1.1 Constant Time Representing Time Symbol

Our first version of the time token t represents a preset time interval specific to the application domain. For example, a sequence of t's between two location terminal symbols represent the temporal property of the node mobility. Suppose that we have three actions A, B and C happening consecutively in an application domain. Then

$$A \;\; 24 \;\; B \;\; 42 \;\; C$$

states that once the event A happens, B happens in 24 time units, and C happens in 42 time units (following B). If the time token is chosen to represent a time interval of 20 time units, the above sequence will be represented (approximately) by the following sentence:

$$A \;\; t \;\; B \;\; t \;\; t \;\; C \;\; .$$

It should be noted that there is a trade-off between the time interval of the time token (resolution) and the complexity of the grammar, which is related to the length of the sentences in the training data. In the above example for instance, if we had set the time interval of the token to be six units, we would have represented the intervals perfectly, but the sentence would have been much longer. This also affects the generalization power of the PCFGs.

2.3.1.2 Time Distribution Representing Time Symbol

Our second version of the time token t represents a Normal distribution of the time that passes between two terminal symbols. Again if we follow the mobility modeling example, a single t (accompanied by the mean and variance values of the distribution) between two location terminal symbols represents the distribution of the time that passes between the movements of a mobile entity between those two locations. Other distributions can be utilized, however, the calculations regarding to finding the empirical sample mean and variance do not change if we utilize a different distribution.

Suppose that similar to the example in the previous section, there are n trips in the observed data, starting with A, then B and C, made by the same or different mobile nodes. However, the maximum likelihood average (from n observations) of the time that takes between A and B is 5.0, and the maximum likelihood variance for the same parameter is 2.0. Similarly, the average for the movement between B and C is 4.0 and the variance is 1.3. Please note that we can assume a Normal distribution for the time that it takes for each movement, and the maximum likelihood mean

(μ) and variance (σ^2) are the parameters required for this distribution. Then the *movement pattern* within the PCFG can be represented as:

$$A\ t_{5.0,2.0}\ B\ t_{4.0,1.3}\ C \quad .$$

Such modeling reduces the size of the grammar, however, it also brings the question of how to calculate these values once the initial grammar has been constructed. During data incorporation stage, single instances of each trip should be kept in memory to calculate the rolling variance value (since the mean will be constantly changing with each new value). Furthermore, we need actual instances as well when we do chunking operations, since the mean and the variance would change. Let us give a simple example to demonstrate what we mean by this. The below grammar provides two types of trips that a mobile node can take with the relative frequencies in parantheses.

$$Start \rightarrow locA\ t_{5.0,1.0}\ locB\ t_{4.0,1.0}\ locC\ (20) \quad |$$

$$locZ\ t_{7.0,8.0}\ locB\ t_{3.0,2.0}\ locC\ (15) \quad .$$

In the above grammar, we see that the mobile node's movement from B to C is a frequent substring (hence a subtrip, one starting after a visit to locA and one after locZ), so it can be represented by a new nonterminal (and its occurences can be replaced by this). While we can easily find the new means from the frequencies (it is $\frac{(20\times4.0)+(15\times3.0)}{20+15}$), the new variance is impossible to calculate since we do not have the original observations. To overcome this, we propose to also keep the sum of squares of all the observations that is in the represented distribution. This way the sample variance can easily be calculated as: $\hat{\sigma^2} = \frac{1}{n}\sum_{j=1}^{n}(t_j - \hat{\mu})^2 = \frac{1}{n}\sum_{j=1}^{n} t_j^2 - \hat{\mu}^2$ (for a given set of n observed time value $t_{1\rightarrow n}$).

Let's give the grammar of the previous example with the new model:

$$Start \rightarrow locA\ t_{5.0,1.0,520.0}\ locB\ t_{4.0,1.0,340.0}\ locC\ (20) \quad |$$

$$locZ\ t_{7.0,8.0,855.0}\ locB\ t_{3.0,2.0,165.0}\ locC\ (15) \quad .$$

Now, if we take the pattern of "locB time locA" as a frequent subtrip and construct the chunk nonterminal, then the above grammar becomes:

$$\begin{aligned} Start \rightarrow & \; locA \;\; t_{5.0,1.0,520.0} \;\; Sub_{BA} \;\; (20) \quad | \\ & \; locZ \;\; t_{7.0,8.0,855.0} \;\; Sub_{BA} \;\; (15) \\ Sub_{BA} \rightarrow & \; locB \;\; t_{3.57,1.67,505.0} \;\; locC \; (35) \quad . \end{aligned}$$

The average, the variance and the sum of squares are calculated as follows. Since the average of the 20 cases from the first rule is 4.0 and from the 15 cases of the second rule of $Start$ is 3.0, then the average for the combined 35 cases is $\frac{(4.0\times20)+(3.0\times15)}{(20+15)=3.57}$. The sum of squares for the 35 cases is just the added value of the sum of squares for the 20 cases of the first rule and the 15 rules of the second rule, hence $340 + 165 = 505$. The variance can be easily calculated afterwards by $\frac{1}{35} \times 505 - (3.57)^2 = 1.67$.

Hence we conclude that keeping the sum of squares is essential to calculate the new averages caused by the chunk and merge operators, and also takes much less space than keeping the separate values for all the cases.

2.3.2 Relative Tokens

Relative tokens represent the change in the state of a system relative to the previous state, which was determined by the previous action (terminal symbol in the sequence). A simple example of such use arises in modeling the movements of a mobile node. In such a case, each new terminal symbol moves a node a constant distance in the direction defined by the relative terminal symbol. See the simple example on Figure 2.4. The node in this example starts at point $A2$, and makes a couple of movements as given in the figure to end up in the square $B6$. Rather than giving all the points it has been to, we only give the relative movements to the previous location of the node. In the figure, U, D, R, UR represents *up*, *down*, *right* and *up-right* (cross movement) respectively.

Employing relative tokens have a couple of advantages. First one is the fact that it decreases the number of distinct terminals, hence the number of nonterminals in the grammar (due to the inference method). This lowers the complexity of the

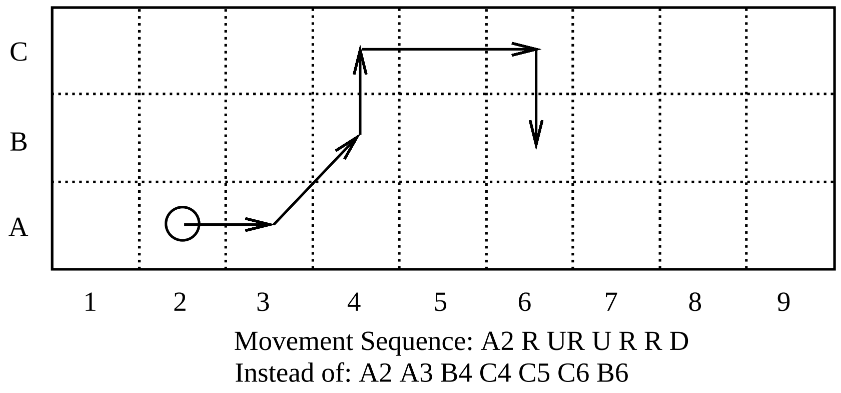

C
B
A
1
2
3
4
5
6
7
8
9
Movement Sequence: A2 R UR U R R D
Instead of: A2 A3 B4 C4 C5 C6 B6

Figure 2.4: Relative Movement Token Example

grammar, and therefore reduces the time it takes to construct the grammar from the training data. Furthermore, it makes the detection of subactions easier by increasing the recurring patterns in the training data, making it easier to find meaningful and frequent chunks. Again in Figure 2.4, 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 sentences by the exact locations, this motion will not be taken as a frequent one; while in the case of relative tokens, a zig-zag subaction will be found as a chunk nonterminal no matter what the exact locations are.

2.4 Improvement over Previous Work

In this section we will describe our improvements over the work presented in [17] and [55]. First of all, the formulation to score how good a grammar is given in both of these papers as the Bayesian posterior probability as:

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

However, this formulation is later replaced in [17], for inference purposes, with the maximization of the posterior probability for the model structure. This means finding the model which gives the highest likelihood given the data when the parameters of this model is assigned using a Dirichlet distribution. The reason for this is given by authors as being a more precise approximation to the Bayes-optimal averaging over the models which represent the grammar. Therefore the search procedure uses the following posterior and related likelihood calculation [17] (of training data, X, given a distribution of left-hand side rule probabilites for the nonterminals, θ_M):

$$P(M_S|X) \propto P(M_S)P(X|M_S)$$

$$and$$

$$P(X|M_S) = \int_{\theta_M} P(\theta_M|M_S)P(X|M_S,\theta_M)\ d\theta_M\ .$$

We however, follow the previous formulation, as also followed in [55], although we apply the operators as given in [17], namely, *merge* and *chunk*. This gives us the following advantage. In the case of the *merge* operation, it is easier to calculate the change in posterior probability, as given in Section 2.2.3.2. This way, we do not have to reparse the sentences that are in the training data, the changes between the previous and the current grammar is enough to calculate the change in $P(D|G)$ (probability of the training data given the grammar). This lowers the calculation of $P(D|G)$ from $O(D^3)$ to $O(D)$ (i.e. from parsing all sentences to looking at only the new rule that is generated by the merge). Furthermore, this lowers the overall complexity of the inference algorithm to $O(D^3)$. We also show in Section 2.2.7 that the constrained search for the merge arguments makes the search for the best merge operation to only $O(D)$.

Our second improvement is in the *chunk* operator. In Section 2.2.2, we introduce a very simple method on the representation of the grammar which directly affects the prior probability, $P(G)$, of the grammar (see Eq. 2.3). This simple method of representation constrains the search for the profitable chunk operation to strings of length up to five, as given in Section 2.2.4, which further lowers down the complexity of this search to $O(D\ log(D))$ from $O(D^2\ log(D))$. As aforementioned, assuming we use the constrained search for the merge arguments, the constrained search for the best chunk argument lowers the overall complexity of the inference algorithm to $O(D^2\ log(D))$.

To summarize, our improvements over previous PCFG inference methods can be listed as:

- 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 time complexity analysis 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).

2.5 Measuring the Effect of Our Approximations to PCFG Inference

In this section, we will give experimental results on how the approximations proposed in Sections 2.2.4 and 2.2.7 affect the process of automated PCFG construction. We will give the time it takes to infer a grammar, and the *a posteriori* probabilities of the generated grammars for differing training data sizes.

For our evaluations, we are utilizing three grammars to generate a varying number of sentences as training data so that the PCFG inference algorithm can process these sets. The grammars we utilize are as follows: (i) $a^n\ b^n$, (ii) $[a^n\ b^n] \mid [c^m\ d^{2m}]$, and (iii) $[a-z]^n$ (i.e. a random string). We compare six types of PCFG inference algorithms (in terms of time it takes to construct a grammar, and $P(G|D)$ of the final grammar, as well as the highest $P(G|D)$ during the inference process) with varying levels of approximation:

- **Approximation Level 0:** *Merge Search* approximation *level-0*, and *Chunk Search* approximation *level-0*,
- **Approximation Level 1:** *Merge Search* approximation *level-0*, and *Chunk Search* approximation *level-1*.
- **Approximation Level 2:** *Merge Search* approximation *level-1*, and *Chunk Search* approximation *level-0*.

- **Approximation Level 3:** *Merge Search* approximation *level-1*, and *Chunk Search* approximation *level-1*.
- **Approximation Level 4:** *Merge Search* approximation *level-2*, and *Chunk Search* approximation *level-0*.
- **Approximation Level 5:** *Merge Search* approximation *level-2*, and *Chunk Search* approximation *level-1*.

Above, *level-0* in *Merge Search* means that all couples 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 couples, however only their effect on each other is evaluated. *level-2* in *Merge Search* applies our highest level of approximation to finding merge nonterminal couples, as presented in Section 2.2.7, 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 length. Again, *level-1* represent our highest level of approximation to find the chunk string, as given in Section 2.2.4, which looks for strings of length up to 5.

For the experiments, we implemented the PCFG inference algorithm and the approximations in *Perl* programming language, due to the ease of implementation. We utilized a very modest system which has an *Intel(R) Core(TM)2 Duo T-8100 2.10 GHz* processor, with 3GB of RAM, running *Windows 7*, to give an idea when we are presenting the results for the time it takes to construct the PCFGs.

2.5.1 Grammar a^n b^n

In our evaluations of the grammar inference approximations, our first grammar consists of a number of a's followed by the same number of b's. To be specific, we initially generated a varying number of sentences from the following PCFG:

$$\text{Start} \rightarrow \text{a Start b (0.9)} \mid \text{a b (0.1)}$$

Windows 7 Enterprise. Copyright © 2009 Microsoft Corporation. All rights reserved. Service Pack 1.

Table 2.1: Final Grammar's A Posteriori Log-Probability (Compared to the Initial Grammar) for All Levels of Approximation in the Grammar a^n b^n Experiment vs Training Data Size (in Sentences)

Training Data Size	Appr. Level 0	Appr. Level 1	Appr. Level 2	Appr. Level 3	Appr. Level 4	Appr. Level 5
200	108.831	199.138	152.27	187.067	161.772	152.446
400	233.861	293.656	207.656	155.251	138.584	158.947
600	188.542	271.1	183.606	171.601	202.307	178.1
800	197.268	116.211	271.264	215.145	167.524	236.272
1000	289.704	237.013	271.624	221.498	252.29	307.915
1200	389.703	280.732	285.589	250.916	329.623	240.056
1400	258.145	236.454	305.14	227.303	235.116	180.405
1600	351.013	208.378	208.981	219.303	292.768	158.01
1800	384.27	242.476	220.396	252.437	299.687	230.232
2000	385.429	292.603	401.881	363.865	302.53	247.389

We later trained PCFGs with varying approximation levels on the set of sentences that were generated by the above grammar. The average length (l_{av}) of a sequence (sentence) generated by this grammar is 20 (i.e. $l_{av} = [0.9 \times (l_{av} + 2)] + [0.1 \times 2]$).

The results of this experiment is given in Tables 2.1, 2.2, and 2.3. We changed the number of sentences between 200-2000 sentences (in steps of 200) for each training data, and we ran 10 cases for each size (the results presented here are the averages) to compare the varying approximation levels of grammatical inference.

Table 2.1 gives the *a posteriori* log-probability results as compared to the initially constructed grammar. As aforementioned, *Sample Incorporation* phase generates the initial grammar (and we take the *a posteriori* probability of this grammar to be β as shown in Algorithm 1), and chunk and merge operators are applied as long the grammar's *a posteriori* does not fall below this initial value. The results provided in this table are the log_2 (logarithm in base 2) of the *a posteriori* difference between the initial grammar and the final grammar (i.e. just before the grammar inference process stops). As it can be observed, in general, the lower approximation in finding operators brings with higher *a posteriori* values. This is an expected result since the heuristic makes better decisions at each step with less approximations.

To look at the *a posteriori* probability metric in another perspective, we check the highest *a posteriori* probability during the inference process in Table 2.2. For

Table 2.2: Best Grammar's A Posteriori Log-Probability (Compared to the Initial Grammar) for All Levels of Approximation in the Grammar a^n b^n Experiment vs Training Data Size (in Sentences)

Training Data Size	Appr. Level 0	Appr. Level 1	Appr. Level 2	Appr. Level 3	Appr. Level 4	Appr. Level 5
200	1291.957	1277.4	1291.7	1277.4	1291.7	1277.4
400	1719.9	1702.7	1719.9	1702.7	1719.9	1702.7
600	1999.474	1976.3	1997.5	1976.3	1997.5	1976.3
800	2270.828	2247.8	2268.5	2247.8	2268.5	2247.8
1000	2452.4	2432.6	2452.4	2432.6	2452.4	2432.6
1200	2622.421	2599.9	2620.1	2599.9	2620.1	2599.9
1400	2601.742	2582.8	2601.7	2582.8	2601.7	2582.8
1600	2827.449	2804.4	2826.8	2804.4	2826.8	2804.4
1800	2729.8	2710.2	2729.8	2710.2	2729.8	2710.2
2000	3160.6	3139.6	3160.6	3139.6	3160.6	3139.6

this metric, we see that the grammar inference algorithms which applies highly approximated methods (proposed in Sections 2.2.4 and 2.2.7) do not perform significantly worse than choosing the operators in a brute-force manner. Although one might argue that the best *a posteriori* probability during the inference process does not allow for enough generalizations (which would lower the *a posteriori* probability); in purely mathematical sense, the grammar with the highest *a posteriori* probability is the "best" grammar, and higher approximation levels also come up with quite good results for this metric.

Finally, Table 2.3 presents the time it took to construct (in average) a grammar for different training data sizes. We see that although the final and best *a posteriori* probabilities do not differ so much (as shown in Tables 2.1 and 2.2), higher approximation schemes provides a tremendously fast way to construct grammar. Another observation that should be done here is that the time it takes to construct the grammars does not increase as drastically as it should with increasing training data size. This is due to the following reason. After the initial construction, the size of the grammars are nearly the same for each training data size, since the pattern is a highly regular one (consisting of only a small set of sentences reappearing all through the training data). Hence, the difference in time that it takes is dependent on the *Sample Incorporation* stage, which still increases (as it can be observed) with

Table 2.3: Average Time (in Seconds) It Takes to Construct the Final Grammar (i.e. Inference Process) for All Levels of Approximation in the Grammar a^n b^n Experiment vs Training Data Size (in Sentences)

Training Data Size	Appr. Level 0	Appr. Level 1	Appr. Level 2	Appr. Level 3	Appr. Level 4	Appr. Level 5
200	13.466	9.52	7.25	1.433	6.753	0.906
400	18.055	11.927	9.223	1.844	8.629	1.187
600	20.825	17.276	10.437	2.373	9.77	1.467
800	27.982	20.637	13.899	2.526	13.07	1.636
1000	31.826	21.738	15.81	2.788	14.902	1.783
1200	31.17	26.458	13.037	2.142	16.607	1.994
1400	31.341	23.633	11.281	2.161	15.123	2.337
1600	33.461	32.288	14.051	2.773	17.197	2.613
1800	31.588	27.395	13.6	2.522	17.454	2.544
2000	41.852	30.22	17.653	4.221	21.883	2.914

the increasing training data size due to the fact that incorporating larger datasets into an initial grammar takes longer, but not as drastically as expected.

2.5.2 Grammar $[a^n\ b^n] \mid [c^m\ d^{2m}]$

Our second PCFG experiments work on the expression $[a^n\ b^n] \mid [c^m\ d^{2m}]$, which generates one of two types of strings: a number of a's followed by the same number of b's, or a number of c's followed by twice of the same number of d's. To be specific, we initially generated a varying number of sentences from the following PCFG:

$$\text{Start} \rightarrow \text{AB } (0.5) \mid \text{CD } (0.5)$$

$$\text{AB} \rightarrow \text{a AB b } (0.8) \mid \text{a b } (0.2)$$

$$\text{CD} \rightarrow \text{c CD d d } (0.8) \mid \text{c d d } (0.2)$$

The average length (l_{av}) of a sequence (sentence) generated by the above grammar is 12.5 (i.e. $l_{av} = (0.5 \times l_1) + (0.5 \times l_2)$ where $l_1 = [0.8 \times (l_1 + 2)] + [0.2 \times 2]$ and $l_2 = [0.8 \times (l_2 + 3)] + [0.2 \times 3]$). Again, we trained PCFGs with varying approximation levels on the set of sentences that were generated by the above grammar.

Similar to Section 2.5.1, we looked at the same three metrics (*a posteriori* log-

Table 2.4: Final Grammar's A Posteriori Log-Probability (Compared to the Initial Grammar) for All Levels of Approximation in the Grammar [a^n b^n] | [c^m d^{2m}] Experiment vs Training Data Size (in Sentences)

Training Data Size	Appr. Level 0	Appr. Level 1	Appr. Level 2	Appr. Level 3	Appr. Level 4	Appr. Level 5
200	84.291	52.733	68.736	61.592	71.617	81.585
400	88.975	54.994	128.377	66.202	110.507	113.578
600	114.345	142.902	129.045	115.171	100.888	149.613
800	158.021	86.888	152.602	169.918	109.907	102.357
1000	163.272	124.98	143.648	140.515	159.317	183.956
1200	149.514	169.892	112.77	180.092	120.341	164.814
1400	171.186	201.313	124.914	177.24	133.348	159.035
1600	195.392	160.302	148.767	160.756	146.528	102.081
1800	149.155	235.616	120.053	195.5	185.97	191.383
2000	200.127	152.654	177.398	179.24	204.37	212.788

Table 2.5: Best Grammar's A Posteriori Log-Probability (Compared to the Initial Grammar) for All Levels of Approximation in the Grammar [a^n b^n] | [c^m d^{2m}] Experiment vs Training Data Size (in Sentences)

Training Data Size	Appr. Level 0	Appr. Level 1	Appr. Level 2	Appr. Level 3	Appr. Level 4	Appr. Level 5
200	589.275	577	589.1	577	589.1	577
400	937.016	920.6	935.8	920.6	935.8	920.6
600	974.5	957.4	974.5	957.4	974.5	957.4
800	1117.706	1101.3	1117.5	1101.3	1117.5	1101.3
1000	1246.541	1223.6	1246	1223.6	1246	1223.6
1200	1358.811	1337	1358.5	1337	1358.5	1337
1400	1422.801	1402	1420.9	1402	1420.9	1402
1600	1574.8	1556.1	1574.8	1556.1	1574.8	1556.1
1800	1651.7	1634.6	1651.7	1634.6	1651.7	1634.6
2000	1638.811	1620.8	1637.7	1620.8	1637.7	1620.8

probabilities of the final and the best grammars, and the time it takes to construct the final grammar, i.e. grammar inference process) in comparing our approximation schemes, and the results are presented in Tables 2.4, 2.5, and 2.6.

Tables 2.4 and 2.5 present the *a posteriori* log-probability difference between the initial and the final as well as the best grammars during the inference process. It can be observed although the best grammar's *a posteriori* is higher for less ap-

Table 2.6: Average Time (in Seconds) It Takes to Construct the Final Grammar (i.e. Inference Process) for All Levels of Approximation in the Grammar [a^n b^n] | [c^m d^{2m}] Experiment vs Training Data Size (in Sentences)

Training Data Size	Appr. Level 0	Appr. Level 1	Appr. Level 2	Appr. Level 3	Appr. Level 4	Appr. Level 5
200	9.908	9.657	2.552	1.229	2.141	0.762
400	16.721	19.732	4.467	2.169	3.86	1.177
600	19.172	19.318	4.581	2.106	3.912	1.263
800	21.687	24.872	5.373	2.232	4.669	1.399
1000	25.243	26.532	6.388	2.523	5.513	1.56
1200	29.326	27.389	7.09	2.689	6.154	1.636
1400	29.999	35.094	7.232	2.937	6.151	1.751
1600	36.134	37.938	8.509	3.186	7.513	1.91
1800	39.648	43.15	9.412	3.768	8.168	2.297
2000	37.833	37.562	9.309	3.56	8.372	2.255

proximation in choosing the operators, interestingly, the final grammar's can be larger for higher approximation schemes. We can attribute this to the fact that the lower approximation schemes can come up with a better merge argument for the last steps which allow for generalization, without going below the initial grammar, but lowering the *a posteriori* probability.

The time that it takes in average to construct the final grammar is given in Table 2.6. Again we see that the approximation brings up tremendous improvement in terms of speed. Furthermore, we see that *Approximation Level 3* (which has *Approximation Level-1* in Merge Search and *Approximation Level-1* in Chunk Search) is faster than *Approximation Level 4* (which has *Approximation Level-2* in Merge Search and *Approximation Level-0* in Chunk Search) which shows the importance of fast discovery of frequent substrings to construct the chunk nonterminal during inference. Finally, again similar to the result in Section 2.5.1, we see an increase in the time with larger training data, but not so drastically, since the grammar is a highly regular one.

2.5.3 Grammar $[a - z]^n$

As aforementioned, the previous two experiments do not really represent the time complexity of the grammar inference task, since the initial grammar (after

Sample Incorporation stage), are more or less the same for each dataset size. In this section, we utilize a random grammar ($[a-z]^n$) to generate a varying number of sentences as training data to be fed to the inference algorithm. With such a randomized scheme for each sentence in the training data, our hope is that the size of the initial grammars constructed will vary due to the differing dataset sizes, and hence the time that it takes to construct the grammar will change significantly. To be exact, the PCFG that we utilized to generate the sentences is as follows:

$$\text{Start} \rightarrow \text{R Start } (0.875) \mid \text{R } (0.125)$$

$$\text{R} \rightarrow \text{a } (0.038) \mid \text{b } (0.038) \mid \ldots \mid \text{z } (0.038)$$

Each sentence from the above grammar generates a sequence of random letters from the alphabet (each 26 have equal probability, i.e. $1/26 \approx 0.038$), and the average length (l_{av}) for such a sequence is 8 (i.e. $l_{av} = [0.875 \times (l_{av} + 1)] + [0.125 \times 1]$).

The results of this experiment are given in Tables 2.7, 2.8, and 2.9. Like the previous experiments, we used the same metrics to compare our approximation schemes. As is expected, the ability to minimize and generalize is minimal in such a randomized set of sentences taken as training data. This behavior can be observed in Tables 2.7 and 2.8 where even for the slightest change in the *a posteriori* probability difference from the initial grammar can happen if the number of sentences is around 100 (this is when there are any advantageous substrings to be taken as chunk operators etc.). It furthermore can easily be seen that the higher approximation schemes perform more or less the same (in terms of *a posteriori* probability) as compared to no approximations (as represented by *Approximation Level* 0) when choosing merge or chunk operators.

In Table 2.9, we present the time results when constructing the grammars, and can observe the drastic change with the increasing training data size. Please notice that we use much fewer sentences (since it takes longer and longer to construct the grammars) than previous experiments and the time values are noticeably higher. For comparison, in the experiment with grammars $a^n\ b^n$ and $[a^n\ b^n] \mid [c^m\ d^{2m}]$, the time it took for 2000 sentences with *Approximation Level* 0 was 41.852 and 37.833

Table 2.7: Final Grammar's A Posteriori Log-Probability (Compared to the Initial Grammar) for All Levels of Approximation in the Random String ($[a\text{-}z]^n$) Experiment vs Training Data Size (in Sentences)

Training Data Size	Appr. Level 0	Appr. Level 1	Appr. Level 2	Appr. Level 3	Appr. Level 4	Appr. Level 5
20	0	0	0	0	0	0
40	0.1	0.1	0.1	0.1	0.1	0.1
60	0.1	0.1	0.1	0.1	0.1	0.1
80	0.9	0.9	0.9	0.9	0.9	0.9
100	2.9	2.9	2.9	2.9	2.9	2.9
120	4.608	4.608	5.108	5.108	5.108	5.108
140	4.236	4.236	4.236	4.236	4.236	4.236
160	4.536	4.536	3.699	3.699	4.099	4.099
180	4.113	4.113	3.6	3.6	3.6	3.6
200	5.208	5.208	3.495	3.495	3.895	3.895

Table 2.8: Best Grammar's A Posteriori Log-Probability (Compared to the Initial Grammar) for All Levels of Approximation in the Random String ($[a\text{-}z]^n$) Experiment vs Training Data Size (in Sentences)

Training Data Size	Appr. Level 0	Appr. Level 1	Appr. Level 2	Appr. Level 3	Appr. Level 4	Appr. Level 5
20	0	0	0	0	0	0
40	0.1	0.1	0.1	0.1	0.1	0.1
60	0.1	0.1	0.1	0.1	0.1	0.1
80	0.9	0.9	0.9	0.9	0.9	0.9
100	2.9	2.9	2.9	2.9	2.9	2.9
120	7.7	7.7	7.7	7.7	7.7	7.7
140	7.6	7.6	7.6	7.6	7.6	7.6
160	19	19	19	19	19	19
180	27	27	27	27	27	27
200	44.5	44.5	44.5	44.5	44.5	44.5

seconds (in average) respectively. For the current experiment the time it takes for the same approximation level is 81.729 seconds (in average) with a training data of only 200 sentences. We furthermore see that the higher approximation levels take much less time (while not giving worse grammars in terms of *a posteriori* probability) to construct a grammar.

Table 2.9: Average Time (in Seconds) It Takes to Construct the Final Grammar (i.e. Inference Process) for All Levels of Approximation in the Random String ([a-z]^n) Experiment vs Training Data Size (in Sentences)

Training Data Size	Appr. Level 0	Appr. Level 1	Appr. Level 2	Appr. Level 3	Appr. Level 4	Appr. Level 5
20	0.023	0.03	0.076	0.042	0.059	0.056
40	0.095	0.092	0.044	0.047	0.09	0.091
60	0.073	0.097	0.123	0.106	0.098	0.079
80	0.227	0.207	0.257	0.292	0.21	0.227
100	0.582	0.556	0.428	0.483	0.412	0.436
120	1.792	1.786	0.926	0.946	0.864	0.963
140	2.376	2.338	1.238	1.189	1.081	1.088
160	14.305	15.581	2.954	2.874	2.417	2.362
180	29.584	28.508	4.076	4.136	3.172	3.143
200	81.729	81.866	7.033	7.03	4.787	4.796

2.6 Conclusions and Future Work

In this chapter, we have presented our approach to the automated construction of probabilistic context free grammars (PCFGs). We have extensively utilized previous work [17], [54], [55] while introducing approximations during the construction to lower time complexity, which in turn helps with applicability of the method to larger datasets. Via time complexity analysis and experiments, we showed that our approximations provide a much faster way to generate PCFGs, while not losing too much in terms of grammar goodness. We have furthermore presented our additions to the PCFGs to make them applicable to specific network application domains, which can be listed as time symbols and relative tokens.

In terms of grammatical inference, we would like to list two directions. The first of these directions is the branching methodology during the construction of the grammar. This can be described briefly as follows. While constructing the grammar, there can be multiple advantageous operations at each step. Rather than choosing the best one, as we currently do, a multiple of those can be kept (with their resulting grammars at each step) to evaluate a larger set of grammars to model the training data in a better way. This has been examined in [17], [54], and it would be very interesting to see the benefit of our approximation schemes in such a methodology.

The second direction is the preprocessing of the training data in terms of frequent substrings. These substrings can be preprovided by the user (hence would be application specific), or still automatically discovered (via the methods provided in *Frequent Sequence Mining*, see Chapter 1). If these frequent substrings are provided by the user, then it means a manual interference with the grammar construction process, however can also be useful since a set of sequences which are meaningful in the application domain provides better modeling of the training data.

CHAPTER 3
PCFGs for Sensor Network Event Recognition

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) collecting 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 [57].

In today's sensor applications that collect vast amounts of measurement data, usually too big for manual inspection, it is very important to provide users with 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. Our purpose in this chapter is to present utilization of PCFGs for modeling the patterns in sensor network measurements that lead to a certain event. We will start the chapter by giving previous work on event recognition in sensor networks. We will next provide a real-world scenario of parking lot monitoring. We will conclude the chapter with the listing of the advantages that can be gained via the usage of the PCFGs in the domain of event recognition in sensor networks. We also list the pitfalls that can occur.

3.1 Related Work for Event Recognition in WSN

We have already listed the many efforts in WSN event recognition via the use of PCFGs when we introduced the PCFGs in Chapter 1. In this section we would like to see some other methods applied in this domain.

The first paper we would like to list [58] presents a two layered event recognition scheme. The initial level detects events from atomic activities using a method similar to frequent itemset mining. The second level uses a type of string matching algorithm to recognize event sequences. Another paper [59] utilizes 3D data maps

* Portions of this chapter previously appeared as: S. C. Geyik and B. K. Szymanski, "Event recognition in sensor networks by means of grammatical inference," in *Proc. IEEE INFOCOM*, 2009, pp. 900−908.

(which can be seen as measurements in 3D space) to recognize events in a sensor network. The spatio-temporal patterns of the time-series data maps are examined to recognize events.

Authors of [60] provide a visual system to recognize user activities, and makes use of Bayesian classifiers as well as finite-state automata to recognize single actor events. For multiple events, a temporal likelihood is calculated for the two possible events occuring together. [61] utilizes fuzzy logic as well as the examination of the sequence of basic operations to recognize events in a resource-constrained sensor network.

The authors of [62] combine first-order logic with the uncertainty of primitive action detection into what is called a Markov Logic Network (MLN) [63]. They apply the MLN methodology to the visual monitoring of a parking lot. A feature extraction method to reduce dimensionality is utilized in [64] to classify and recognize human actions in a distributed way, within the domain of a wearable motion sensor network.

Markov Models are also frequently applied in literature on event detection [65]-[67]. [65] utilizes hidden Markov Models (HMM), to recognize human actions from a time-series of images. Their experiments consist of recognizing six types of tennis strokes. The paper in [66] makes use of coupled HMMs (CHMM) [68] to recognize actions which are performed by 2 or 3 people. [67] compares the HMMs and CHMMs on recognizing human behaviors in a visual recognition system, and concludes with CHMM being the superior method.

3.2 Real world Scenario and Simulations

Following [51], 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. Figure 3.1 shows a car tracking example and tracking done to gather symbols from the camera image.

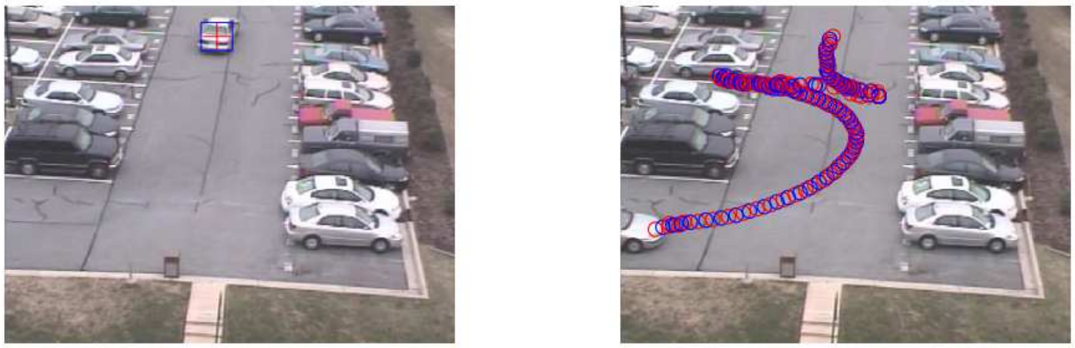


Figure 3.1: Parking Lot Car Tracking for Event Detection

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 event made by a car (regardless if this is the most recently entered or some other car). 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.

Figure 3.2a and Figure 3.2b show the car trajectories associated with different events in the parking lot simulation. Squares with the letter P 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*).

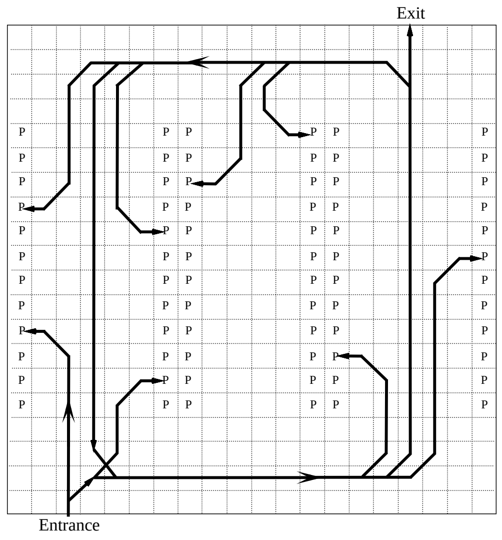
First two of these events and their corresponding actions are shown in Figure 3.2a. Note that a car can circle around the lot for a few times to find a parking spot before leaving. In the simulations, we assumed that a car leaves the parking lot after an unsuccessful circle with probability of 60% and starts another circle around the parking lot with probability of 40%.

In Figure 3.2b, 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.

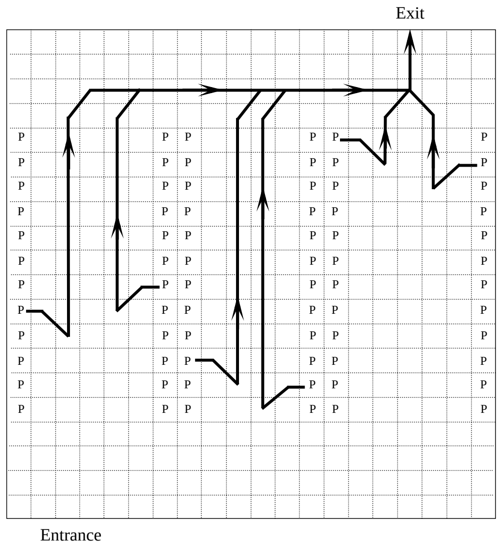
We have run the training data generation program (for feeding into grammar inference algorithm) with above given parameters for 100000 time units. Size of training data acquired for each event is: *Enter and Park (1468 sentences, 25333 symbols)*, *Leave Parking (1396 sentences, 17698 symbols)* and *Enter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

Figure 3.3, Figure 3.4 and Figure 3.5 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 more backward actions)*, *tr (turn right 45 degrees)*, *tl (turn left 45 degrees)* and *s (stop)*. Clearly, movement tokens are relative tokens described in Section 2.3.2. Furthermore, we combine multiple forward and



(a) Car trajectories for the events: *Enter and Park* and *Enter and Leave*



(b) Car Trajectories for the event *Leave Parking*

Figure 3.2: Car Trajectories for the Real-World Scenario

backward movements into a single terminal symbol (f and b) to simplify preprocessing, taking advantage of the fact that a forward or a backward movement does not change direction.

The grammars fully model the actions of a car representing events shown in Figure 3.2a and Figure 3.2b. 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. 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 Figure 3.3). Careful examination reveals that nonterminal *C3* reduces to $4 \times (f\ tl\ f\ tl)$ 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. 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 the string gets longer). In the grammar for the event *Enter and Leave*,

```
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)
```

Figure 3.3: PCFG for Event Enter and Park

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 Figure 3.5.

3.3 The Advantages and Disadvantages of PCFGs for the WSN Domain

Probably the biggest advantage of using PCFGs for detecting events in sensor networks is the ability to utilize recursive grammar rules for repeating action sequences. Our real-world scenario provides a perfect example where a car can make an infinite number of circles around a parking lot with decreasing probability, to find an empty parking spot. Such a relation is very difficult, if not impossible, to capture with pure k-level history based models (such as Markov models). Further-

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)

Figure 3.4: 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)

Figure 3.5: PCFG for Event Enter and Leave

more, PCFGs have the capability to infer subevents too. The scenario provides also such an example where a nonterminal actually presents the subevent of a car's single circle around the parking lot.

Unlike Markov models, PCFGs are perfect for capturing long term relationships between symbols that occur in an event sequence. This discussion can be based on automata theory also; while a Markov model can be interpreted as a probabilistic finite state automaton (with no stack), a PCFG is a probabilistic pushdown automaton (with stack). It can be stated that PCFGs can recognize a larger set of languages, hence have higher modeling capabilities for events.

An issue that occurs while using PCFGs is the problem of finding the time window for the occuring event. A Markov model has a predefined window and hence there is no further processing to detect the start and the end of an event to recognize it. However, such a problem can be overcome in PCFGs by setting up time limits for an event's length, and working in a window spanning a certain time period.

Markov models are comparably simpler than PCFGs and it is easier to infer transitions as well as transition probabilities of a Markov model than the production rules and rule probabilities of a PCFG. A smaller data set is often sufficient to recognize what type of primitives can follow what primitives and usually give a good approximation of these transitions' probabilities in a Markov model. A PCFG's production rules are usually complex and have many variations, which usually require a larger data set to fully infer. Event recognition domain is relatively easy to collect data of ordinary real-life cases. For example, it is easy to acquire a large data set of examples of what a normal action sequence for a person in an airport is. The same is valid for the parking lot monitoring application where a camera can be deployed to view a parking lot and it can monitor normal parking sequences for months, producing a very large data set. However, the extreme cases such as a security breach in an airport does not occur commonly in real life and such cases must be manually constructed by humans, hence bringing the problem of insufficient size data sets. For such cases, a rough idea of what the dangerous patterns are could be implemented by a synthetic data generator, and many cases with small variations can be created; however the synthetic cases are not usually good indicators of what a real-life danger would be like. We can conclude that the normal operation of a system is much more easier to capture by a grammar (later used to detect abnormalities), and given its expressivity benefits, it is better to use grammars for such applications. If we would like to model abnormalities, it is more beneficial to use simpler models, e.g. Markov models, due to the problem of insufficient amount of data that can be collected in the event recognition domain.

3.4 Conclusions and Future Work

In this chapter, we have presented the applicability of PCFGs in the domain of sensor network event recognition. We have shown results based on synthetic data, which demonstrated the generalization capabilities of PCFGs in this domain, and promise for future utilization. We have also listed the advantages and disadvantages that the PCFG-based event recognition may bring. As a future work direction in this domain, we can list the application of PCFG methodology on real-life event data (which may be noisy, hence require further processing) and evaluate, or improve, the performance of PCFGs in such settings.

CHAPTER 4
Mobility Modeling and Synthetic Trace Generation with PCFGs

Mobility of nodes is one of the key attributes of today's networks. The examples of networks in which nodes move around most of the time and use wireless communication are mobile ad hoc networks, delay tolerant networks, robotic networks and mobile sensor networks.

New protocols and algorithms for wireless mobile networks benefit from their verification via simulations in their early design stages. However, such simulations require large amount of realistic mobility behavior data, which are difficult to collect. Therefore, development of methods which can generate long synthetic mobility data from sample traces is crucial for proper evaluation of protocols and applications via simulation.

In this chapter, we present our efforts in modeling the movements of nodes in a mobile network through the automated construction of PCFGs. 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. 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

* Portions of this chapter previously appeared as: S. C. Geyik, E. Bulut, and B. K. Szymanski, "PCFG based synthetic mobility trace generation," in *Proc. IEEE Globecom*, 2010, pp. 1–5.

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 chapter, this is not the case for PCFGs. 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.

The rest of the chapter is as follows. We first provide previous work on mobility modeling. Next, we discuss *Mobility PCFGs* in Section 4.2, which enable the grammar to model both spatial and temporal properties of the mobile node movements. This is followed by the synthetic mobility trace generation algorithm which utilizes the mobility grammars in Section 4.3. Finally, we evaluate the PCFG-based mobility modeling method. First, we demonstrate the benefits of PCFG-based mobility modeling compared to other models in terms of describing entity movements in Section 4.4. Next, we provide the evaluation of how close the generated traces are to the original trace and we compare our method with a generator based on Markov Models in Section 4.5. We also compare the complexity of building a PCFG versus building a Markov Model, and discussing if the performance increase in modeling accuracy makes up for the added time complexity by the PCFGs. We conclude the chapter and present future work in Section 4.6.

4.1 Literature Review of Mobility Pattern Modeling and Trace Generation

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 [70], Manhattan [71] etc., even though real domains often contain obstacles or preset paths. Recent works that address these challenges include [72]-[74]. In [72], anchor points are used to describe how a mobile entity can move in an environment with obstacles. Thus, each anchor point acts as a guide for routes in the area that avoid hitting obstacles. [73] 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 [74]. It takes into account the locations visited and the movement preferences of different mobile nodes. The authors also propose to use time periodicity where the movement preferences change in time, and certain preferences periodically reappear.

There have also been many attempts at creating synthetic mobility patterns based on traditional random movement mobility models, including methods based on connectivity graphs [75], action profiles [76], terrain and vehicle properties [77], group behavior [78] and events [79]-[81].

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 [82], [83]. A time-variant community mobility model is proposed in [84]. 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 [85]. They generate mobility traces based on pedestrian densities on streets as well as likely paths that the pedestrians take. Both [84] and [85] make use of real world traces to build their models for generating synthetic traces. In [86], 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, [87] 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 [88], transitions between areas are modeled by their probabilities. Markov Model based mobility predictors are compared to LZ-based mobility predictors in [89] 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 [21]. A 2-level Markov Model is used to predict connectivity and quality of connection to access points in a mobile network in [22].

We compared our PCFG-based synthetic data generation to a 2-level Markov Model based generator presented in [21]. 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. Both PCFG and Markov Models hold more information than classical statistics based approaches. 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.

4.2 Mobility PCFGs

The original definition of the PCFGs is not sufficient to fully encapsulate the mobility properties of nodes. To capture spatial patterns of node movements, a PCFG can be built 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 Chapter 2 (see Section 2.3.1). Furthermore, depending on the complexity of the mobility modeling task (size of the area, variety on the movements etc.), relative tokens (see Section 2.3.2 in Chapter 2) can also be used.

Algorithm 2 Method for creating a random route for a mobile node from the mobility PCFG given an initial location of this mobile node

```
$init\_loc$ =initial location
$g$ =mobility grammar
for each rule $r$ in $g.START$ do
   $string = r$
   for each expansion $string_i$ of $string$ with terminal symbol at position 0 do
      if $string_i[0] == init\_loc$ then
         $list.add(string_i)$
      else
         $delete(string_i)$
      end if
   end for
end for
normalize probabilities in $list$
$random = rand()$
$progressive = 0$
for all expansions $e_i$ of every string in $list$ do
   $progressive + = prob(e_i)$
   if $progressive \geq random$ then
      return $e_i$
   end if
end for
```

4.3 Synthetic Trace Generation of Node Mobilities with PCFGs

Synthetic trace generation is basically creating a sentence from the constructed grammar. This sentence gives both the temporal and spatial information for the single mobile node. 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. Hence, we present a single algorithm here, which gets as input the mobility grammar and initial location of the mobile node (can be *null* for a node that has just begun its journey), and creates a new sequence beginning in the initial location. Of course, the probabilities of the production rules are taken into account when deciding which rule to apply next in sentence generation process. The silent assumption here is that the input data contain traces starting at a location that is also the ending location of some trace.

In Algorithm 2, the initial stage checks for all possible movement sequences (hence all possible sentences produced by the PCFG), and keeps only the ones in which the first terminal symbol is the same as the initial location of the mobile node. In the case of modeling meetings of mobile nodes, the symbols are the mobile nodes met, hence although the algorithm stays the same, the meanings of the symbols produced or matched are different. After the initial elimination, the remaining productions are chosen according to a probability distribution. Here, since the sum of all productions before elimination (but not after it) is 1.0, a normalization is done by multiplying accordingly all the branches of parsing tree of selected sentences.

4.4 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.

4.4.1 Expressing Other Mobility Models as PCFGs

There have been 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.

4.4.1.1 Random-Waypoint to PCFG Transformation

A mobile node adhering to the random-waypoint mobility model [70] moves in steps. In each step, it chooses a random point in the mobility area and then gets there with a random velocity. 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 velocity between the values floor (f) and the ceiling (c). The continuous location and velocity generation is performed iteratively. In this setting, the random-waypoint grammar can be represented by the following PCFG:

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

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

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

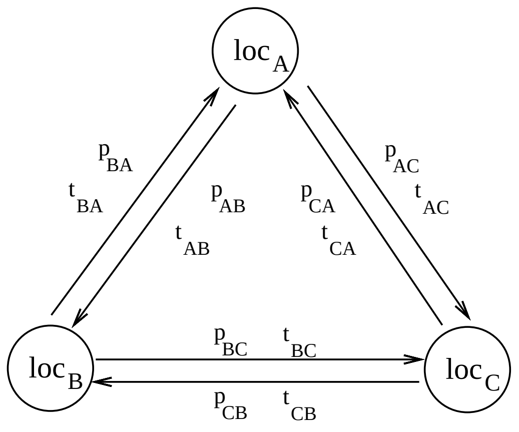
Thus, at each *Start* production, the above grammar decides on a new location and the corresponding node moves there with the random velocity generated in the previous production. The first production of the nonterminal determines the initial location of the mobile node. The length of the mobility period 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.

4.4.1.2 Markovian Mobility Model to PCFG Transformation

Markov model based mobility models [21], [88], [89] 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 4.1 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 takes for a node 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 nonterminals 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, and then does the transition (as the nonterminal of the new 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).

4.4.2 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



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}

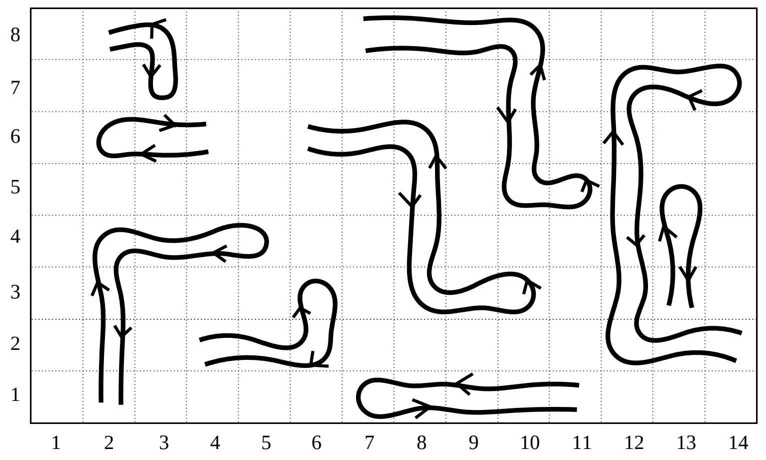
$T_{CA} \rightarrow$ Time amount following distribution t_{CA}

$T_{CB} \rightarrow$ Time amount following distribution t_{CB}

Figure 4.1: A Markovian Mobility Model and Its PCFG Transformation

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) [90], Prediction by Partial Matching (PPM) [91], and LZ-Based [92], [93] 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 Chapter 2 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 Chapter 3). Furthermore, these methodologies mainly deal with the task of prediction, whereas our application of PCFGs in this paper works with



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

Figure 4.2: A Palindromic Mobility Example and the PCFG that Describes It

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. 4.5). 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.

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 Non-deterministic Push-down Automata [94], 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 4.2, we give a mobility example, which is actually very feasible in a real-world application. 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. 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.

4.5 Evaluation of the Trace Generation Method

In this section we evaluate the PCFG-based mobility generation method to a Markov Model based generator as given in [21]. We first give how the mobility properties are preserved by both methods. The last part of the section compares the time complexity of constructing the mobility PCFG from a given real-world trace to the complexity of building a Markov Model from the same trace.

4.5.1 Evaluation of Mobility Modeling Accuracy

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 [95] 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 [96]. To train the PCFG for [95], 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 [96], we have divided

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.)

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

the area into 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 $\Delta_{Cons} = \sqrt{\sum_{i=1}^{s}(r_i \, . \, (g_i - r_i))^2}$ (where s is the number of meeting sequences of length k).

Table 4.1: 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 $\Delta_{Intern} = \sqrt{\sum_{i=1}^{p}\sum_{s=1}^{r}(w_{i,s} \, . \, (tg_{i,s} - tr_{i,s}))^2}$ (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 Chapter 2. Table 4.1 provides a description of what we mean by *Appr. Level k*. These were mentioned in Chapter 2 as well, but we would like to repeat these here for clarity. 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 2.2.7, 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 2.2.4, which looks for strings of length up to 5.

The results for the DieselNet Dataset [95] are presented in Table 4.2. 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 4.2, although the grammars take longer to construct and generate synthetic data, they provide traces that are much closer (up to 93%) to

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 4.2: Evaluation Results for DieselNet Dataset

	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	0.108
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 2	0.002	0.002	0.002	0.002	0.002	0.002	0.018
Cons 3	0.003	0.002	0.002	0.002	0.002	0.002	0.016
Cons 4	0.004	0.0033	0.003	0.002	0.003	0.003	0.03
Cons 5	0.005	0.004	0.003	0.003	0.003	0.003	0.041
Cons 6	0.004	0.003	0.003	0.003	0.003	0.003	0.041
Intern 2	1.8	1.45	1.47	1.26	1.26	1.28	11.46
Intern 3	3.06	2.56	1.6	1.95	1.88	1.76	9.40
Intern 4	4.74	3.23	1.73	1.89	1.87	2	40.53
Intern 5	5.2	3.84	1.91	1.79	1.98	2	52.62
Intern 6	4.88	3.72	1.82	1.67	1.85	1.68	60.03

Table 4.3: Evaluation Results for the First 500 Routes in Taxi Mobility Dataset

	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 2	0.008	0.019	0.005	0.005	0.005	0.004	0.099
Cons 3	0.015	0.033	0.006	0.007	0.007	0.007	0.043
Cons 4	0.017	0.024	0.007	0.008	0.008	0.008	0.058
Cons 5	0.018	0.021	0.008	0.01	0.009	0.009	0.071
Cons 6	0.02	0.022	0.009	0.011	0.011	0.011	0.08
Intern 2	46.45	64.12	8.81	8.92	12.37	10.39	58.45
Intern 3	82.76	123.07	14.95	11.98	16.98	16.55	74.74
Intern 4	121.61	136.58	14.5	13.01	16.25	14.51	220.83
Intern 5	151.48	150.01	12.04	10.53	12.15	14.14	269.92
Intern 6	128.6	92.24	12.17	10.89	11.24	14.54	290.57

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* 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 [96], 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 [96] are shown in Table 4.3. 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 4.3), and for the whole dataset (Table 4.4), 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 4.4: Evaluation Results for the Whole Taxi Mobility Dataset

	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

4.5.2 Complexity Comparison between PCFGs and Markov Models

In Section 2, we have shown the time complexity of our PCFG inference algorithm to be $O(D^2log(D))$, where D is the size of the training data that we construct the PCFG from (due to at most $O(D)$ applications of the chunk operator which costs $O(D\ log(D))$ with the search). Furthermore, we concluded that the space complexity is $O(D)$, since the space that the model is kept in does not grow larger after the initial storing of the training data, which naturally takes $O(D)$ space. Now we are going to do the similar evaluation for the Markov Models.

The space complexity of a Markov Model is easy to calculate. Please note that what a simple Markov Chain of preset length holds is a set of patterns with the distribution of the next symbol to come, i.e. for mobility modeling, two previous location sets with a distribution on which location will be next. One can see that for a discretized map of the area, the following locations should be neighbors to each other, and this way there can be at most n_{neigh}^{k+1} set of patterns that are being held. n_{neigh} is the maximum number of neighbors that a mobile entity can go to from one location, and k is the level of the Markov model, i.e. the history that matters. This value can be taken to be $O(1)$ since it does not change with the size of the training data (and n_{neigh} is constant). However, for a generalized Markov Model, outside of the domain of mobility modeling, even if we always come across different patterns, the space complexity is $O(D)$, since there can not be more patterns than the size of the training data.

For the time complexity, it is trivial to see that training a Markov Chain of preset depth (a k-level Markov chain) is like searching for a *chunk* operator in our PCFG inference case. Patterns of length $k+1$ are held with their frequency of appearance, which later are used to calculate transition probabilities. Hence, the time complexity to construct a general k-level Markov Chain is $O(D\ log(D))$. However, as aforementioned, in the case of mobility modeling, since there can be a maximum number of different patterns, this complexity falls down to $O(D)$, since we no longer search for a location in the hash table (to keep the frequency of a pattern being seen) in time which is related to the size of the training data. One can also argue that even in the case of PCFGs, this mobility restriction makes the search for the chunk operator take $O(D)$ time, hence the overall time complexity of automatically constructing a PCFG becomes $O(D^2)$. In both cases, we can conclude that the inference of a PCFG is an $O(D)$ times more costly operation that the construction of a k-level Markov Chain, in terms of time complexity. In the case of Hidden Markov Models (HMM), a well known training algorithm, *Baum-Welch* [14], exists which is a special case of an *Expectation Maximization* algorithm. This algorithm iterates over the parameters of the HMM, and changes them at each iteration to *locally* maximize the likelihood of the current parameters given the training data. The time complexity of this algorithm for each iteration is $O(S^2D)$, where S is the number of hidden states, and D is again the size of the training data. While one can argue that HMMs are not necessary in our current example of mobility modeling, in other domains the decision on the number of hidden states and the number of iterations play a vital part in the complexity of this algorithm.

The trade-off between the representational accuracy and the complexity of PCFGs and the Markov Models should be taken into account for different application domains, since both has advantages. We go over these advantages in the other chapters of this thesis as well.

4.6 Conclusions and Future Work

In this chapter, we have focused on the usefulness of PCFG framework for modeling mobility properties of nodes in data collected by or generated by a network. We provided evaluations based on real world traces for thus domain. 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.

Our future work in this domain includes the evaluation of the synthetic traces generated by the PCFG in terms of network metrics, such as congestion, load, latency etc. As aforementioned, our PCFG-based mobility modeling provides realistic generation of movements for mobile nodes, hence the experiments performed on PCFG-generated mobility traces should give different results as compared to different mobility models (random movement-based or others). For this purpose we are aiming to employ multiple datasets, which may arise from social environments as well (hence taking into account the movements of entities due to their relationships to others).

CHAPTER 5
Behavior Modeling in Social Networks with PCFGs

Social networks analysis [97] includes examining the actions of entities in a social setting. These actions can be either interactions between entities (e.g. talking, exchanging items etc.), or actions which do not include interactions, but nevertheless in the social context, hence resulting from the social setting. Such actions often contain patterns which are specific to the application domain. For example, it is natural to assume that a brainstorming session often include group discussions which are not dominated by one person but rather involves dyadic communications. In the eyes of a single entity however, this application domain means that the opposing person (hence the interacting entity) changes rapidly.

It is important to understand these patterns to realize the characteristics of a social setting. A simple example can be given in a company where there is a certain order in the interactions. If a customer has to go through certain department tables to accomplish a certain task, the placement of these departments in the commonly used order saves time and energy for both the customer and the worker. Another example can be given in the military context. A large community has a certain set of interaction patterns between its members which can be observed in the long run. A potential dangerous activity can be recognized from the abnormal patterns that occur in a community that is being monitored. Such a recognition ability can also be used in the work environment to predict failures as well as to improve performance in a business process.

In this chapter, we introduce a novel method for modeling social node behaviors through the utilization of PCFGs. Given a set of action sequences in a network setting, an automatic generation method, which we already presented in Chapter 2, can build a PCFG which concisely holds probabilities for action patterns. This PCFG can either be used to predict or classify future behaviors, or to understand

* Portions of this chapter previously appeared as: S. C. Geyik, J. Xie, and B. Szymanski, "Behavior modeling with probabilistic context free grammars," in *Proc. Int. Conf. Information Fusion*, 2010, pp. 1−8.

these patterns due to its concise nature and appropriateness for manual inspection.

The rest of the chapter is as follows. In the next section, we discuss the recent developments in behavior modeling subdomain in social network analysis. In Section 5.2, we give the results of our PCFG modeling on the Mission Survival Corpus 1 (MSC-1) dataset [99] owned by Project FBK (Fondazione Bruno Kessler). We provide grammars constructed for socio and task label categories defined within the dataset. Furthermore, we present the classification results based on the grammars acquired from this dataset. Following the results, we finalize the paper with the conclusions and future work in Section 5.3. We also discuss the usefulness of the PCFG approach compared to previously applied schemes to the domain of social network analysis.

5.1 Behavior Modeling Efforts in Social Network Analysis

Let us now provide a short literature survey listing previous behavior modeling work. A large body of research aims at modeling *physical* interaction phenomena, such as utterances, gestures, eye gaze, head position and body motion, from multimodal sensor streams. Instead of providing straight-forward semantic meaning of the behavior, most of these outcomes serve as higher level features for algorithms attempting to infer semantic interactions between individuals. For example, [100] discussed an approach to estimate who was talking to whom based on head positions of the participants. [101] used gazes, head gestures, and utterances to determine interactions regarding who responds to whom in multiparty conversations. [102] proposed a classifier to recognize agreement or disagreement utterances, utilizing both word-based and prosodic cues. In [103], audio and body motion features are used to assess interest and attraction level in conversational dyads.

By taking the individual interaction patterns into account, there are quite a few papers focusing on recognizing the group activities as a whole. [104] investigated a statistical framework for automatic meeting analysis based on various hidden Markov models. A set of meeting actions (including monologue, presentation, white-board etc.) are defined based on turn-taking patterns. A range of audio-visual features (e.g. speech activity, pitch, speaking rate, head and hand blobs) from each participant are extracted and combined to investigate the group-wise nature of the actions. More recent work is presented in [105] by utilizing a two-layer HMM framework. [106] introduced a dynamic Bayesian network based model, characterized by multiple stream processing to segment the group states similar to [104], but with features including prosody from audio signals, speaker activity and lexical features from textual transcripts. [107] defined the group states as discussion, presentation and briefing. A decision tree classifier was trained to estimate group states using four frequency-based features (including number of overlaps in speech, number of changes of speakers, number of participants who had spoken and average length of overlaps). By defining the states as presentation, discussion and break, [108] presented event-driven multilevel dynamic Bayesian networks (DBNs) to perform online detection of multilevel events, which coordinates both the multicue-based bottom-up reasoning and context-based top-down guidance.

In addition to modeling through interactions, the work on automatically identifying the roles (including functional, social and dominating) in a group has drawn a lot of attention. [109] used Bayesian methods to recognize functional roles (e.g., the anchorman, the second anchorman, the guest) in broadcast data. [110] recognized roles in movies (e.g., hero), based on social network analysis. By combining both the lexical feature and the social networks describing the interactions between the meeting participants, [111] discussed an approach to identifying the project manager, the marketing expert, the user interface expert and the industrial designer. [112] introduced a machine learning approach based on multiclass SVMs with radial based kernels to recognize roles such as giver, seeker and recorder in meetings by employing audio-visual features. [113] investigated a new framework for functional role detection based on the *influence model* [114], which takes into account the features of other participants. [115] proposed an DBN based approach for discovering influence in a lounge where people played interactive debating games, which automatically determines how much influence a member has on others on a pair-wise basis. [116] and [117] described models for automatically detecting the team members who play a dominating role in a meeting using both support vector machines and a dynamic Bayesian network with a two-level structure.

Different from the above-mentioned methods, we present the utilization of grammars to model actions based on the social role and the interactions between group members, which provides a straightforward but more detailed interpretation of the group behaviors.

5.2 PCFG-based Social Entity Behavior Modeling

We have used Mission Survival Corpus 1 (MSC-1) dataset [99] to show the applicability of our methods. Basically, we model what type of behavior is followed by a person that undertakes a certain social or task-functional role. For each role, we have automatically generated the grammar of the interactions of the holder of this role with others. Our analysis of the PCFG-modeling on this dataset is two-fold. First, due to their conciseness, we present the resulting grammars in this section and present our manual inspection on them, which provides important information about roles that the social entitites undertake. The second part of our analysis provides classification results. We first provide how the grammars constructed from the entire dataset performs in separating the role classes. Later, we utilize part of the dataset as training data and test on the rest of the data, where we evaluate how good the grammars perform in classifying the unseen actions of social entities into the roles they undertake.

The MSC-1 dataset basically contains time-stamped annotations of 11 meetings of people deciding on how to proceed in a disaster scenario. The annotation includes the *socio role label* and *task role label* of each meeting attendant as well as an indication of who speaks at the time-stamp. Socio roles (*supporter*, *protagonist*, *attacker*, *neutral*) mainly represent the attendant's attitude towards the group's function while the task roles (*giver*, *seeker*, *orienteer*, *neutral*) represent the individual's function and technical skills. Although the role names are self-explanatory, the interested reader can look up detailed role descriptions in [99]. Furthermore, we will give brief explanations while we are presenting our results.

In the MSC-1 dataset, we modeled two sets of action patterns (which we also call the first and second *metric*):

- **Which roles go together:** Given a social role (socio label) of an attendant, what task roles are taken (sequence-wise) by this attendant during this social role? Or vice versa.

- **Attendants of which roles are speaking while a certain role is undertaken:** When an attendant of a meeting plays a certain social role, what are the social roles of the other attendants that speak (hence we assume, interactions while the attendant undertakes this role) during this role? The same question is valid for the task roles.

The next subsection provides our manual inspection of the grammars acquired for both metrics. The other subsection of this section provides our classification results.

5.2.1 Manual Inspection of Grammars

Let us present the grammars constructed for the MSC-1 dataset. The modeling results for the first metric (*Which roles go together*) can be seen in the grammars in Figures 5.1-5.4. The results for the second metric (*Who Speaks while a Certain Role is Undertaken*) are presented in Figures 5.5-5.8.

START $\rightarrow$
C4 N2 (0.01)
N3 (0.08)
N2 (0.18)
C1 (0.06)
N0 C2 (0.01)
C4 N3 (0.02)
C2 (0.06)
N0 M1 (0.01)
M1 (0.03)
N0 N2 (0.14)
N0 (0.19)
N0 N3 (0.07)
C4 (0.14)

N0 $\rightarrow$
n (1)

N2 $\rightarrow$
g (1)

N3 $\rightarrow$
o (1)

C1 $\rightarrow$
N0 C4 (1)

C2 $\rightarrow$
N3 N0 (1)

C4 $\rightarrow$
N2 N0 (1)

M1 $\rightarrow$
M1 N0 (0.31)
s (0.69)

Figure 5.1: PCFG for Modeling Which Roles Go Together for Supporter Socio Label

Figure 5.1 gives the grammar constructed to model first metric (*Which roles go together*) for the *supporter* socio label. The supporter has a cooperative attitude and, furthermore, provides resource support to other attendants. This mentality is clear in the given grammar. Orienteer (*o*) and giver (*g*) terminals represent mainly the supporting attitude of the attendant. Among these, giver is probability-wise the most dominant task-label that can be seen in the grammar. It can also be observed that seeker (*s* in the nonterminal *M1*) is another task-label that can appear in the supporter interaction sequence. This basically represents the cooperative attitude, since the conversation (cooperation) between two individuals in a meeting proceeds by questions (*s - seeker terminal*) and answers (*g - giver terminal*).

```
START→          |  N0 →
N0 M1 (0.18)    |  n (1)
M1 (0.18)       | ---------
M1 N0 (0.55)    |  M1→
N0 (0.09)       |  s (0.5)
                |  g (0.5)
```

Figure 5.2: PCFG for Modeling Which Roles Go Together for Attacker Socio Label

The grammar that models first metric for the *attacker* socio label can be seen in Figure 5.2. As the name suggests, the attacker *attacks* the opinion of another attendant. Such a behavior can be definitely observed in the grammar since both seeker (terminal *s*) and giver (terminal *g*) has equal probability (as given by *M1*). Such information can be easily observed thanks to the concise representation provided by the PCFG that is automatically constructed from the dataset.

```
START→           |  C3 (0.06)   |  C2 →
N0 N1 (0.13)     | ---------    |  M1 N0 (1)
C1 (0.12)        |  N0 →        | ---------
M1 (0.07)        |  n (1)       |  C3 →
N0 (0.34)        | ---------    |  N0 C1 (1)
N0 C2 (0.04)     |  N1 →        | ---------
C2 (0.06)        |  p (1)       |  M1→
N1 (0.14)        | ---------    |  a (0.04)
N0 M1 (0.04)     |  C1 →        |  s (0.96)
                 |  N1 N0 (1)   |
```

Figure 5.3: PCFG for Modeling Which Roles Go Together for Giver Task Label

Next, let us examine the grammar in Figure 5.3 constructed for modeling the first metric for *giver* task label. A giver basically provides factual information. By the nonterminal *M1*, a supporter (s) is much more dominant than an attacker (a) in this situation. The probability of a giver being a protagonist (the attendant that drives the conversation, denoted by p in the grammars) is quite high (∼40%). Please note that although such results can be obtained through statistical means, grammars provide much more compact and manually examinable complete information about the meeting data. Furthermore, certain grammar processing algorithms (e.g. backward chaining from the terminal to the *START* nonterminal) can be utilized to estimate the probabilities with which certain terminals occur in the sentences as well as the transition probabilities of symbols [54]. This way, such analysis of behaviors from the grammars does not have to be manual at all. Some extra information about the giver grammar is the fact that protagonist and attacker or supporter do not ever occur together. This means that the protagonist socio-label does not usually change its opinion and then becomes the supporter or attacker of another person. Instead, it is rather he who is attacked or supported. Furthermore, an attacking or supporting giver does not try to take control of the conversation. The grammar's power comes into attention in such situations, because such comments require the examination of sequences.

START→
N0 M1 N0 (0.02)
N0 M1 (0.06)
M1 (0.08)
M1 N0 (0.05)
N0 (0.52)
N1 N0 (0.07)
N0 N1 (0.07)
N1 (0.13)

N0 →
n (1)

N1 →
p (1)

M1→
a (0.22)
s (0.78)

Figure 5.4: PCFG for Modeling Which Roles Go Together for Seeker Task Label

Figure 5.4 shows the grammar that models the first metric for the *seeker* task label. Seeker asks questions. However, from nonterminal *M1* (created by a merge operation), we see that a person asking a question is more likely to be a supporter (s) than an attacker (a). The most dominant socio-label is the neutral (n). The next one is a supporter who is the main socio-role taken by seekers. It is also remarkable any significant socio-label (i.e. not neutral) is either followed or preceded (mostly) by the neutral socio-label. This shows two things. For the proceeding case, we see that seeker either gets satisfied or his supporting status ends after his confirmation question. For the preceding case however, it can be concluded that for the annotators (the people that annotated this data by manually observing attendants), it takes a while to understand whether a person asking question is a supporter or attacker. Also, it can be seen that the seeker could be a person who answers his own questions, or drives the conversation by his questions, hence acting as the protagonist (p).

START →
N13 (0.03)
C3 N13 (0.03)
N8 N4 N17 (0.03)
C16 (0.03)
C13 (0.10)
C4 N13 (0.03)
N17 C1 (0.03)
N9 N10 (0.03)
N9 N17 (0.06)
N9 N13 (0.03)
C14 N17 (0.03)
C6 C1 (0.07)
C2 M1 N8 N17 (0.03)
C1 (0.10)
C13 N17 (0.07)
N9 (0.20)
N17 (0.10)

N0 →
n-s (1)

N4 →
p-p (1)

N8 →
n-p (1)

N9 →
n (1)

N10 →
n-n (1)

N13 →
s (1)

N17 →
p (1)

C1 →
N9 C14 (1)

C2 →
N17 N8 (1)

C3 →
N13 N0 (1)

C4 →
N13 N9 (1)

C6 →
C1 N10 (1)

C13 →
N17 N9 (1)

C14 →
N10 N9 (1)

C16 →
C6 N9 (1)

M1 →
n-n-p (0.7)
n-p-p (0.3)

Figure 5.5: PCFG for Modeling Who Speaks while a Certain Role is Undertaken for Supporter Socio Label

Using the grammar in Figure 5.5, let's examine who speaks while an attendant undertakes the socio label role of *supporter*. As expected of a supporter, the other attendants that speak while a supporter plays his role is mostly protagonists (p). For example, from the *START* nonterminal, it can be seen that *N17* is a really (probabilistically) dominant nonterminal which points to a single speaker, protagonist (p). We can see that there can be multiple protagonists speaking at the same time, given by the nonterminal *M1* (e.g. *n-p-p* means a neutral and two protagonists speaking at the same time). However, even from that nonterminal, not looking at the rules of *START* nonterminal, we can see that it has a lower probability as compared to a single protagonist. This is given by the probabilities of rules being 0.3 to 0.7 in the nonterminal *M1*.

```
START→
N11 N4 N3 N11 N4 N3 N11 N8 N11 (0.09)
M1 N9 N1 N3 N10 N3 N1 N8 (0.09)
N1 N3 N1 (0.09)
C1 C1 C1 (0.09)
N8 (0.09)
N8 N11 N8 N1 (0.09)
N1 N6 N1 N0 N6 (0.09)
N1 (0.09)
N9 N10 N9 N1 N9 (0.09)
N8 M1 N6 N8 (0.10)
C1 C1 N8 (0.09)
---------
N0 →
n-s (1)

N1 →
n (1)
---------
N3 →
n-n (1)
---------
N4 →
n-n-p (1)
---------
N5 →
p-p (1)
---------
N6 →
s (1)

N8 →
p (1)
---------
N9 →
a-n (1)
---------
N10→
a-n-n (1)
---------
N11→
n-p (1)
---------
C1 →
N8 N5 (1)
---------
M1→
a (0.5)
p-s (0.5)
```

Figure 5.6: PCFG for Modeling Who Speaks while a Certain Role is Undertaken for Attacker Socio Label

Figure 5.6 shows the grammar that models the second metric (*Attendants of which roles are speaking while a certain role is undertaken*) for the *attacker* socio label. It is worthwhile to notice in the attacker's grammar that it is either another attacker (*a*) or a protagonist (*p*) who speaks. A protagonist can speak with a supporter (*s*) at the same time (as in nonterminal *M1*). However, we do not see a supporter and an attacker speaking at the same time. From this information, we can say that arguments take turns. What may be even more interesting is the fact that we do not see a protagonist and another attacker speaking in the same sentence. This is an important finding which strengthens our deduction that people with opposite ideas take turns.

For Figure 5.7, we can briefly state that the only significant speaker during a *giver*'s turn is another giver, this means a cooperative information providing. While constructing the grammars, we remove the rules with a probability below 0.001, which we assumed are insignificant. In the original (uncut) grammar, we were able to see a seeker (*s*) as well as orienteer (*o*) during the role of a giver in the meeting. These rules were removed when we set the threshold for presentation purposes.

START →
C12 (0.05)
C28 (0.19)
N5 (0.34)
N10 (0.24)
C3 (0.09)
C31 (0.09)

N5 →
g (1)

N10 →
n (1)

N12 →
n-n (1)

C3 →
N5 N10 (1)

C4 →
N12 N10 (1)

C12 →
N10 C4 (1)

C28 →
N10 N5 (1)

C31 →
C3 N5 (1)

Figure 5.7: PCFG for Modeling Who Speaks while a Certain Role is Undertaken for Giver Task Label

Finally, the second metric grammar for the *seeker* task label is given in Figure 5.8. A seeker asks questions; hence it is only natural to see that these questions are satisfied by a giver (g) who provides factual information. This can also be seen in the grammar as the only significant terminals are including givers. Among the deleted rules (original grammar), we were able to see orienteers and seekers (since an orienteer can also contribute to solving problems by organizing meetings, and a seeker can be the further questioner), but these were removed in the thresholding process.

START →
N8 (0.25)
C3 M1 (0.07)
C1 N8 (0.06)
M1 (0.06)
N4 (0.28)
C9 C2 (0.06)
N4 C5 (0.06)
C4 (0.10)
C1 (0.06)

N4 →
g (1)

N5 →
g-n (1)

N8 →
n (1)

N10 →
n-n (1)

N15 →
g-g (1)

C1 →
N4 N5 (1)

C2 →
N15 N4 (1)

C3 →
N8 N10 (1)

C4 →
N8 N4 (1)

C5 →
C8 N4 (1)

C8 →
N8 N5 (1)

C9 →
N4 C2 (1)

M1 →
C3 N8 (0.78)
n-n-n (0.22)

Figure 5.8: PCFG for Modeling Who Speaks while a Certain Role is Undertaken for Seeker Task Label

5.2.2 Classification Results

In this section we will provide the classification abilities of our PCFG-based modeling approach on the MSC-1 dataset [99]. For these purposes we utilize the first metric (i.e. *Which Roles Go Together*) to classify the roles of meeting attendants. We first provide the separation capability of the PCFGs between roles when we utilize the whole dataset; and then, we separate the dataset into training and test partitions and evaluate the classification ability of PCFGs on the unseen data.

Table 5.1 presents the classification results for the social roles in MSC-1 dataset. As previously mentioned, the social roles in the dataset are *attacker* (attacks the opinion of another meeting attendant), *neutral* (a neutral stance), *protagonist* (attendant drives the discussion), and *supporter* (supports the opinion of another meeting attendant). We model the task role sequence performed by an attendant during the undertaking of a social role (as aforementioned for the metric *Which Roles Go Together*), and classification for a sequence into the four social roles is done as follows. First, four grammars (belonging to four social roles) are constructed automatically, and then we try to parse the sequence that is to be classified by these four grammars. Similar to a *Naive Bayesian Classifier*, the classification metric is calculated as the multiplication of the sequence's production probability given a grammar, and the prior probability of the class itself (according to how frequently it is observed in the training data). To account for unseen sequences in the training data, we inserted a *smoothing* nonterminal into each grammar (which can produce any symbol seen so far in the training data), where any nonterminal which generates a terminal also points to with a very small probability (hence effectively can replace the terminal it mainly produces with any other terminal during parsing or production, but this production branch will have a very small probability). This way, we account for replacement errors, as well as prevent the parsing failure (hence assignment of zero probability) of future sequences that are still in the class that is represented by this grammar, but just were not in the training data.

Let us examine Table 5.1. The column *Separation Accuracy* denotes how well the sequences in the dataset were classified after the grammars were constructed with again the whole dataset. Hence both the training and the test data are the

Table 5.1: Separation and Classification Results on the Metric Which Roles Go Together for the Social Roles in MSC-1 Dataset

Between the Roles of	Separation Accuracy	2-Fold Cross Val. Classification Accuracy	10-Fold Cross Val. Classification Accuracy
Protagonist and Supporter	69.7 %	65.4 %	65.3 %
Neutral and Supporter	71.4 %	69.2 %	66.6 %
Neutral and Protagonist	63.5 %	63.6 %	60.3 %
Attacker and Supporter	94.3 %	92.6 %	93.7 %
Attacker and Protagonist	96.1 %	95.4 %	95.8 %
Attacker and Neutral	97.3 %	97 %	96.9 %
Neutral, Protagonist, and Supporter	51.7 %	51.1 %	48 %
Attacker, Protagonist, and Supporter	68.2 %	63.7 %	63.7 %
Attacker, Neutral, and Supporter	70.1 %	67.8 %	65.2 %
Attacker, Neutral, and Protagonist	62.6 %	62.5 %	59.2 %
Attacker, Neutral, Protagonist, and Supporter	51.1 %	50.4 %	47.3 %

same and equal to the MSC-1 dataset (i.e. the processed version where the dataset is transformed into a set of sequences for each role). We have removed the action role sequences consisting of only a single *neutral* symbol (i.e. the sequence "*n*") from the test data, since an attendant that performs no action gives no clues to his/her social role. The columns *2-Fold Cross Val. Accuracy* (Val. is for Validation), and *10-Fold Cross Val. Accuracy* provide the classification results where the training data and test data are separate. To be exact, *2-Fold Cross Val. Accuracy* has 50% of the dataset as training, and 50% of the dataset as test data (an average of two batches is taken, hence we first use the first half as training and second half as test data, and then the second half as training and the first half as test data). *10-Fold Cross Val. Accuracy* is similar to *2-Fold Cross Val. Accuracy*, where an average of 10 batches is presented, where, for each batch, a different 90% of the dataset is used as training data, and the rest 10% is used as the test data.

From Table 5.1, it can easily be observed that the *attacker* social role is easily distinguished from the rest of the social roles, while the others are harder to

Table 5.2: Separation and Classification Results on the Metric Which Roles Go Together for the Task Roles in MSC-1 Dataset

Between the Roles of	Separation Accuracy	2-Fold Cross Val. Classification Accuracy	10-Fold Cross Val. Classification Accuracy
Orienteer and Seeker	63.6 %	66.9 %	67.7 %
Neutral and Seeker	88 %	87.8 %	86.7 %
Neutral and Orienteer	84.4 %	82.1 %	81.3 %
Giver and Seeker	87.9 %	87.2 %	87.4 %
Giver and Orienteer	83.8 %	81.9 %	81.8 %
Giver and Neutral	59.2 %	57.2 %	55.5 %
Neutral, Orienteer, and Seeker	75.8 %	73.5 %	72.4 %
Giver, Orienteer, and Seeker	75.3 %	72.9 %	73.1 %
Giver, Neutral, and Seeker	55.3 %	53.4 %	51.8 %
Giver, Neutral, and Orienteer	54.1 %	51.5 %	49.6 %
Giver, Neutral, Orienteer, and Seeker	50.8 %	48.4 %	46.6 %

distinguish according to their action role sequences. We also see that the 10 and 2-fold cross validation results are not much worse than testing on the training data itself, which demonstrates the classification ability of the PCFG model on unseen data. Furthermore, the classification always performs better than chance, even when classifying between all the four social role classes simultaneously.

Table 5.2 presents the classification results for the task roles in MSC-1 dataset [99]. As aforementioned, there are four task roles in the MSC-1 dataset: *giver* (provides facts), *neutral* (no action), *orienteer* (attendant organizes the discussion), and *seeker* (the attendant asks questions). Under the metric *Which Roles Go Together*, we model the social role sequence undertaken by an attendant while performing a specific task role. The classification experiments as well as the description of the columns in Table 5.2 is similar to the ones we have for Table 5.1, hence we do not repeat them here. From the table, it can be observed that the task role classes *neutral* and *giver* are easy to distinguish from the task role classes *seeker* and *orienteer* in terms of their social role taking, but the classification performs worse between the classes in these two groups (i.e. *neutral* is difficult to distinguish from *giver*, and *seeker* is difficult to distinguish from *orienteer*). The reason why the classification results for groups that include both *giver* and *neutral* are poorer than other groups is that the frequency of the classes *giver* and *neutral* are higher, hence the poor classification performance between those two classes dominates the overall classification performance. Finally, we can again observe that the PCFG-based classification always performs better than chance, even when classifying between all the four task role classes simultaneously.

5.3 Conclusions and Future Work

In this chapter, we have proposed the use of PCFGs to model actions in the social network context. We demonstrated this methodology by modeling social interactions as well as role taking in Mission Survival Corpus 1 [99]. From the modeling results, we can conclude that grammars are a concise way of storing social action patterns and help with inferences about the application domain's social properties. As can be seen, the models we gathered were concise and manageable enough to even manually detect a set of key properties on the behavioral patterns of meeting attendants. We have also provided classification results of PCFG-modeling on the same dataset in separating social entities undertaking different roles. The initial results are promising as to the applicability of the PCFG-based classification in this domain. The main contribution of the PCFG modeling, as compared to other models, is that it includes all the data in a compressed form, hence not losing information. Furthermore, PCFG is a formal model which can be processed by easy modifications to the parsing algorithms, such as Earley-Stolcke [17], [47].

Future work directions that can be listed here include various other application domains within social network context where the PCFGs can be used for prediction and classification, such as performance evaluation of business processes. Furthermore, although it may affect the context-freeness of the PCFG approach, domain specific annotations, that can be integrated into the PCFG model, may come up with a better representation of social network environments.

CHAPTER 6
Service Composition in Sensor Networks and Utilization of PCFGs

Service modeling and service composition are software architecture paradigms that have been used extensively in web services where there is an abundance of resources. They mainly capture the idea that advanced functionality can be realized by combining a set of primitive services provided by the system. Many efforts in web services domain focused on detecting the initial composition, which is then followed for the rest of service operation. In sensor networks however, communication among nodes is error-prone and unreliable, while sensor nodes have constrained resources. This dynamic environment requires a continuous adaptation of the composite service model. The contents of this chapter can be listed as follows. We will first provide our contributions in the area of service composition in sensor networks. We have proposed a novel method for modeling a service, as well as heuristics to efficiently combine services in sensor networks [119]. After building the foundations of the subject as our published work, we will give our work on modeling compositions in sensor networks, and learning of these compositions via the PCFG construction mechanism. We will present the methodology as well as simulation-based evaluations of our method. We will conclude the chapter with our recent efforts in switch options based selection of services in sensor networks and operating modes for per-

* Portions of this chapter previously appeared as, and are to appear in:
S. C. Geyik, B. K. Szymanski, P. Zerfos, and D. Verma, "Dynamic composition of services in sensor networks," in *Proc. IEEE Int. Conf. Services Computing*, 2010, pp. 242−249.
J. Wright et al., "A model-driven approach to the construction, composition and analysis of services on sensor networks," presented at the Annu. Conf. ITA, London, UK, 2010.
B. K. Szymanski, S. Y. Shah, S. Geyik, S. Das, M. Chhabra, and P. Zerfos, "Market mechanisms for value of information driven resource allocation in sensor networks," in *Proc. IEEE Int. Percom Workshop Information Quality and Quality of Service for Pervasive Computing*, 2011, pp. 62−67.
S. Geyik, B. Szymanski, P. Zerfos, and A. Mowshowitz, "Sensor service selection through switch options," in *Proc. IEEE Int. Conf. Service Computing*, 2011, pp. 717−724.
S. C. Geyik, S. Y. Shah, B. K. Szymanski, S. Das, and P. Zerfos, "Market mechanisms for resource allocation in pervasive sensor applications," *Elsevier Pervasive Mobile Comput. J.*, vol. 8, no. 3, pp. 346−357, Jun. 2012.
S. Geyik, E. Bulut, and B. Szymanski, "Utilizing PCFGs for modeling and learning service compositions in sensor networks," in *Proc. IEEE Int. Conf. Service Computing*, 2012 (In Press).

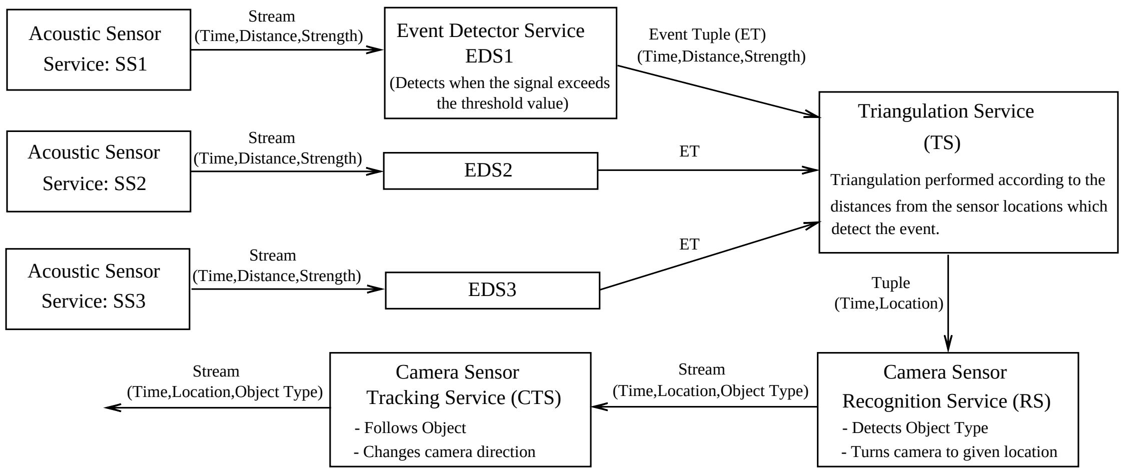

Acoustic Sensor Service: SS1
Acoustic Sensor Service: SS2
Acoustic Sensor Service: SS3
Stream (Time,Distance,Strength)
Event Detector Service EDS1 (Detects when the signal exceeds the threshold value)
EDS2
EDS3
Event Tuple (ET) (Time,Distance,Strength)
ET
Triangulation Service (TS)
Triangulation performed according to the distances from the sensor locations which detect the event.
Tuple (Time,Location)
Camera Sensor Recognition Service (RS)
- Detects Object Type
- Turns camera to given location
Stream (Time,Location,Object Type)
Camera Sensor Tracking Service (CTS)
- Follows Object
- Changes camera direction

Figure 6.1: A Composite Service Example

vasive applications.

6.1 Service Composition in Sensor Networks

Due to limited communication bandwidth, node processing and energy resources, sensor network applications are distributed over a collection of nodes. Each node typically provides a basic functionality for operating on the monitored data, while the network of sensor nodes collectively provides a composite service (i.e. a service that is formed through a suitable combination of basic functionalities [120]) to the end-user.

As an example of a composite service, consider the tracking and object identification application of Figure 6.1: audio measurements are collected from three acoustic sensors and are used to localize the source of the sound. The localization information is then transmitted to a camera sensor that identifies the type of the object that produces the sound. The camera is further used for tracking the object as it roams in the monitored field. In this example, one can readily identify the primitive functionalities that are collectively used to provide the more sophisticated tracking and object identification service.

Service composition, i.e. the process through which composite services are produced by combining several primitive ones [121], has been the subject of extensive

In this chapter, we use the terms *application* and *service* interchangeably.

study in the context of web services [120]. However, the unique characteristics of sensor networks render techniques that were devised for web service composition inadequate. Unlike the web environment where service provider availability and ample communication bandwidth is typically assured, sensor networks are highly dynamic as nodes often fail or become disconnected and wireless communication capacity is limited. Thus, web service composition approaches (e.g. [122]-[127]) are susceptible to single-point-of-failure and inefficient in the use of precious wireless communication bandwidth.

Additionally, the service paradigm exhibits qualitative differences in the web and sensor network domains: in the web, service consumers are typically concerned with finding service instances from a plurality of web providers that can accomplish a given, abstract task or functionality [128], [129]. In sensor network deployments, the service paradigm is primarily concerned with the assembly of data transformation pipelines over data flows. In Section 6.1.1, we discuss important implications that this distinction in the service model have on the optimization of the service composition process.

Early programming frameworks [130], [131] proposed for the development of sensor services recognized the need for a component-based design that compartmentalizes (at the source-code level) the transformational steps that network collected data must undertake. Recent advances in sensor network programming further extend this concept by proposing the use of a high-level language such as Haskell [132], WaveScript [133] or Prolog [134] to describe the interconnection of application components, each of which is implemented in a lower-level, device-specific language. However, such programming models are also not robust enough for the dynamics of sensor network environments, neither do they make efficient use of the limited network resources. There is no provisioning for flexible re-engineering of the sensor application at *runtime* should the nodes that provide the services fail or become disconnected and the service model is not adapted to the ever-changing processing and energy resources of the nodes.

The objective of the work presented in this section is to introduce a modeling and composition framework for sensor services that is robust to sensor node and communication failures and efficient in the use of the underlying resources.

In this service-oriented approach, sensor network applications are represented and viewed as a collection of component services assembled in a data flow graph that describes the composite service. Each component service provides basic operators for transforming the data, has typed inputs and outputs, and generates metadata that provides meta-information on the values that are being transformed, as well as on the runtime properties of the sensor nodes that implement the service. Such runtime information may include the cost of processing data at each node and transferring it between nodes in the network.

The graph-based modeling of sensor services along with the cost information are used to formulate the process of dynamic sensor service composition as a cost-optimization problem, which we further prove to be NP-complete. We devised two heuristic methods to solve this problem, which differ on how the composition process proceeds: the top-down approach starts with the high-level specification of the desired composite service and proceeds in multiple steps of refinement by identifying the primitive component services that can be used to provide its inputs. In the bottom-up approach, the service graph is topologically sorted, and each service waits for its candidate input provider services to decide on their own compositions, then chooses a subset of them to satisfy its own inputs. The motivation behind the top-down approach stems from the need to provide cost-efficient service composition graphs due to the limited resources in the network, while the bottom-up approach is devised due to the requirement for robustness in the face of sensor node and communication failures. Centralized and distributed implementations were evaluated for both approaches. In summary, the work presented in this section makes the following contributions:

- a modeling framework for sensor services that follows a data flow graph formulation, which is amenable to analysis,
- a formulation of the dynamic sensor service composition process as a cost-optimization problem, which is shown to be NP-complete,
- two algorithms that use heuristics for solving the dynamic service composition

problem, along with an analysis of their complexity, and

- evaluation of the algorithms using numerical and ns-2 simulations to measure their costs and robustness to node failures.

The rest of this section is organized as follows. Section 6.1.1 outlines a model of sensor services and formulates the composition process in sensor networks. Section 6.1.2 describes our two approaches to sensor service composition, namely the top-down and the bottom-up, along with their centralized and distributed implementations. Simulation results are provided in Section 6.1.4. Section 6.1.5 discusses related work on modeling and composition of web and sensor services. We present our conclusion and future work in Section 6.1.6.

6.1.1 Sensor Service Modeling and Composition

In this section, we first motivate the need for a new modeling and composition framework for sensor network services. We then introduce a new model and describe a formulation of the composition process within that model as a cost-optimization problem, which we further show to be NP-complete.

6.1.1.1 Motivation

Service modeling and composition have been extensively studied for web services and business processes over a number of years [120], [135]. A number of standards [136]-[138] have been adopted and widely used in real-world deployments that developed languages and tools for describing web services, enabling automatic discovery, composition, enactment, and monitoring of web services. However, in the sensor network domain, both the unique challenges of the operating environment as well as the data-driven approach of communication call for a rethinking of the services paradigm.

To begin, the limited resources of sensor nodes (caused by energy depletion from batteries , constrained wireless communication bandwidth, low processing capabilities, etc.) make heavyweight web service modeling languages, protocols and frameworks such as WSDL [137], BPEL [136], and SOAP [139] inapplicable for sensor networks. For example, while there exists the notion of cost in web services [124], the communication costs incurred through the interaction of the component services, which is expressed either in terms of number of messages exchanged or consumed bandwidth, is often not modeled explicitly, as it is not typically a concern in that domain. Furthermore, as the levels of resources continuously fluctuate in sensor network deployments, the composite service needs to dynamically adapt to these changing conditions.

Secondly, while web services are commonly assumed to be always available as they are provided by always-on servers and robust cloud infrastructures, node failures and communication disruptions in sensor networks are to be expected and do occur frequently. Any service composition approach that depends on a single centralized planner node is prone to such failures, making the process of composing services less robust. Considering only costs of communication (as in recent efforts [126]) without taking into account the possibility for node failures is not sufficient for providing fault-tolerant composition in these environments. To avoid single-point-of-failures and increase fault-tolerance in the presence of faulty nodes, it is desirable that the service composition method executes in a distributed manner.

Thirdly, in the web services and business processes case, the service model follows a process- (or workflow-) oriented paradigm, whereas sensor applications implement a data-driven model. One important implication of this fundamental difference lies in the way service composition (and the assorted cost-optimization) approach evolves: in the former case, composition involves the binding (assignment) of an abstract service model and its tasks to service component instances [122], [128]. The latter case is concerned with (potentially partial) matching of the input data requirements of a service to another's output data streams, without possessing an explicit composite service model *a priori*. Consequently, in web services, QoS-measures at the local level can be optimized in polynomial time, while for the sensor environment, as we prove later in Section 6.1.1.7, optimization at even the local service selection level is NP-hard.

In summary, the dynamically changing (and limited) resources, the frequent node failures, and the data-driven notion of services in the sensor environment call for a service composition approach that is cost-efficient, fault-tolerant and suitable for data-driven applications. In what follows, we present such modeling and composition framework that possesses these characteristics.

6.1.1.2 Modeling Sensor Network Services

A service s_i in a sensor network is defined by the input data that it accepts, the transformation function that it applies on its input, the output data that it produces, as well as metadata that provide additional information that characterizes the service and its outputs:

$$s_i = \{input_i = (input_{i,1}, ..., input_{i,m}),$$

$$output_i = (output_{i,1}, ..., output_{i,k}),$$

$$f_i(input_i) \rightarrow (output_i), metadata_i(t)\} \ .$$

Following the above definition, a sensor network, in a service-oriented sense, can be defined as a set of services, abstracted from the sensor nodes and base station(s) that form it:

$$S = \{s_1, s_2, ..., s_n\}.$$

It is apparent that a service implemented in a sensor network may be just a *source service*; i.e., a service which does not receive any input and only outputs data. Furthermore, we can also define a *sink service*; i.e., a service which does not output anything and just receives input. In an application, the end-user requesting information is usually represented as a sink service. In the service definition given above, *metadata* is the information on the service's characteristics; i.e., it is the information shared between services, which gives the properties of the data that are produced by the service such as levels of reliability, period, etc.

Metadata may also include cost information and certain characteristics of the service itself, as well as of its inputs and outputs, such as energy consumption per output data produced, processing delays, number of other services that make use of its outputs, etc. The metadata of a service depends on time (t), due to the dynamic

It is possible that the output of the end-user service is needed for higher level services, in which case it is not a sink but an intermediate node in this higher level service.

conditions of the underlying sensor network environment. Each specific service has separate metadata that are transmitted to other services offered by the sensor network. Metadata information is used in our dynamic composition algorithms to find which services are most cost-efficient to use for a given composite service requested by the end-user. Interested readers can check [140] for an approach to the modeling of metadata in the service oriented architecture (SOA) for sensor networks, which we also follow. Alternative approaches, e.g. ontologies, for encoding metadata might also be applicable, however their exact application details is beyond our scope of study.

6.1.1.3 Service Graph of a Sensor Network

Service graph of a sensor network, G_S, consists of vertices representing services and of directional edges representing possible flow of data between the services. The edge directed from the vertex of service A to the vertex of service B is created if and only if the output of A and input of B intersect in some fields. That is, informally, A can provide some of the data that it generates (output) for use by the service B through this directional edge. A formal definition of the service graph is given below:

$$G_S = \{V, E\} \;\; where,$$

$$V = \{s_i\} \text{ (one vertex for each service) and,}$$

$$E \subseteq V\text{x}V, \; where \; e_{i,j} = \begin{cases} 1 & \text{if } (output_i \cap input_j) \neq \emptyset, \\ 0 & \text{otherwise.} \end{cases}$$

Note that, although not stated explicitly in the graph definition, two services can have an edge between them if and only if the metadata of the service providing some data as output match with those of the service that is requesting this data as input, in addition to having compatible input/output. For example, if a service requires input with reliability of at least a certain level, only the services that can provide this type of data at the requested reliability level can potentially be connected to the service description with a directed edge. While the above definition of the

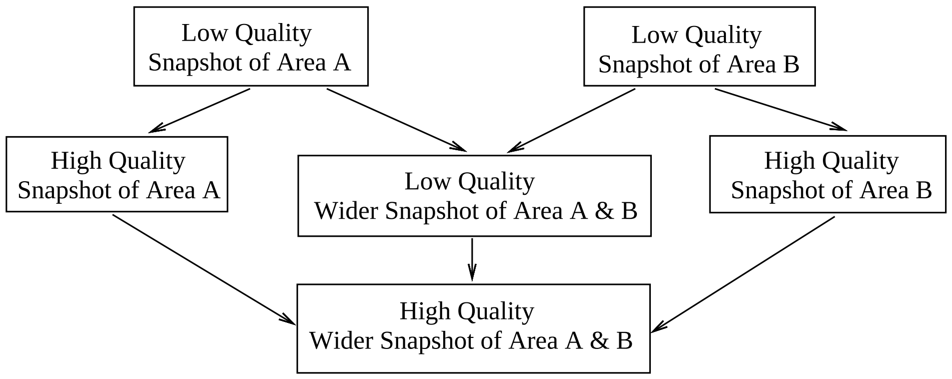

Low Quality Snapshot of Area A
Low Quality Snapshot of Area B
High Quality Snapshot of Area A
Low Quality Wider Snapshot of Area A & B
High Quality Snapshot of Area B
High Quality Wider Snapshot of Area A & B

Figure 6.2: A Simple Type Hierarchy

service graph seems to require exact match between input and output fields of two services to build an edge between them, this requirement can be relaxed by using type hierarchies for the data fields, as further described in the following section. It is worth noting that in case one service can provide input to another, the actual flow of data is a responsibility of the underlying network and routing layers. However, through the use of metadata, the service model (and the composition algorithms) is informed of the communication costs involved when such data flow occurs.

6.1.1.4 Type Hierarchy of Data Fields

Figure 6.2 shows a simple example of a type hierarchy, regarding the snapshot (image) collected by a camera sensor that monitors a certain area. There are two areas (A and B) and two quality levels (high and low) for a picture. In the graph that describes the type hierarchy, it can be observed that the information type *High Quality Snapshot of Area A* is a *subtype* of *Low Quality Snapshot of Area A*, since it already includes the information that is contained within the latter type. Another example can be given for *High Quality Wider Snapshot of Area A & B*, which includes the information for both areas A and B, hence it is a subtype of both *High Quality Snapshot of Area A* and *High Quality Snapshot of Area B.* More elaborate type hierarchies can be built for application specific purposes.

Such a type hierarchy enables two services to be linked in the service graph G_S, even though their outputs and inputs might not match exactly. If a service A's output field is a subtype of a service B's input field, then this means that A can provide the information that B requires and possibly more (what is referred to as “subsumption-based similarity” in [127]). While more complex descriptions of hierarchy using formal logics might be applicable, we strive for simplicity and a light-weight approach in our target environment, and do not consider them in this study.

Furthermore, type hierarchies can be used for replacement of services in an already constructed composition graph for a composite service S. In such a composition, a service A can be used to substitute another service B, if the following conditions hold: a) each input field of A is a *supertype* (opposite of subtype) of each corresponding input field of B, and b) each output field of A is a subtype of each matching output field of B (hence service A requires less information to run, and produces more information than B). Such a service substitution will give the same functionality as the original composition scheme, as long as the properties of metadata are also satisfied.

6.1.1.5 Cost Formulation of Service Composition in Sensor Networks

There are two basic types of costs related to a composition: first, the processing cost of each service, i.e., cost incurred by activating an instantiation of a service. Second, the cost of communication between two services by exchanging information. This cost is interpreted as the edge costs in the service graph defined in the previous section. Note that while we refer to such values in an abstract way as costs, different types of real costs are represented. For example, processing cost of a service can be the energy spent by the sensor node that provides this service, the delay that is incurred by this service, etc. The same holds for the communication cost between services. In that case, we are actually using the cost values that are defined by the underlying structure of the network. For example, energy spent by sending information from service A to B includes all the energy spent by the nodes on the route from A to B. Calculation of such underlying costs is related to the mapping from the sensor network topology to the service graph. Therefore, a simplified way of computing this cost could be to find the shortest path between two nodes in the sensor network and put this value between the services on these two nodes in the service graph. We discuss this subject further in Section 6.1.2.6. Note that it is also possible to define a cost vector, which accounts for multiple costs incurred at the same time. Then, compositions can be ordered according to this cost vector, by using appropriate weights for the cost types.

6.1.1.6 Problem Definition

Service composition requires finding such a set of services $S_C \subset S$ and data flows between those services so that every service in S_C has its inputs provided by at least one other service in S_C. Furthermore, the union of the outputs of services in S_C must satisfy a user-requested functionality Φ, given as a set of output fields required by the end-user satisfying certain properties:

$$\Phi = \{output_{\Phi,1}, ..., output_{\Phi,n}\}.$$

Service composition may be considered as the task of finding a subgraph of the service graph (G_S) defined in the previous section, wherein only a subset of the possible edges (data flows) and vertices (services) is used. Moreover, this problem requires that the cost of the composition is minimized. A formal definition of the problem is as follows: For given $G_S = \{V, E\}$ and Φ, find the minimum cost $V_C \subset V$ and $E_C \subset E$, such that,

$$\Phi \subset \bigcup_{V_i \in V_C} (output\ of\ V_i) \ \text{ and,}$$

$$\forall V_i \in V_C, (input\ of\ V_i) \subset \bigcup_{V_j\ where\ e_{j,i} \in E_C} (output\ of\ V_j).$$

As it can be seen, an edge $e_{i,j}$ cannot be chosen in the composition scheme unless both V_i (representing service i) and V_j (representing service j) are selected for the composition. According to this problem formulation, the composition process may change dynamically, since an optimal composition is dependent on the network conditions at time t. Services learn the network conditions via the metadata mechanism.

6.1.1.7 Service Composition Problem is NP-complete

In a simple version of the above problem formulation of sensor service composition, we have a set of source services, which only give output, a user-request Φ (a sink service) and we wish to find a subset of these source services so that their output fields will satisfy the input fields of the user-request, while keeping the service cost below a certain level. For purposes of illustration, we assume that this cost is additive, such as the total energy consumed by the services that are used. The service composition problem is more general than the above formulation, but even this restricted version is NP-complete by a simple polynomial transformation from the set cover problem. The set cover problem accepts a set, and a set of subsets of this set with a cost assigned to each subset. The problem is to find a subgroup of these subsets so that the original set is covered and cost is below a given value. As is well-known [141], the set cover problem is NP-complete.

Theorem 1. *Service composition is NP-complete.*

Proof. We will transform set cover problem to service composition. For any set cover instance, let S_{cov} denote the set to be covered and for each $i \leq 2^{|S_{cov}|}$, let Sub_i denote a subset of S_{cov}. For each Sub_i, we are given the cost c_i and the problem is to find a cover S_{cov} with cost smaller than c_{req}. The required transformation is as follows:

- Transform S_{cov} to be Φ, the user-request with the required input as the set to be covered,
- Transform each Sub_i into a service with no input, and output being the same as Sub_i, cost of running this service is c_i.

After such a transformation, a procedure finding the composition with the cost lower than c_{req} will also solve the set cover problem. Therefore the service composition problem is NP-hard. Next we will prove that it is in NP.

Given a composition solution (chosen services and data flow), it takes linear time with input to see that the composition satisfies all the input requirements of the services activated in it and the end-user requirements. The complexity of checking the cost of the solution is $O(|V| + |E|)$, since each edge in the solution will only be checked once and each vertex will only be traversed once. Indeed, each edge may incur a cost of transmission and each service incurs a cost of processing. Since we can check the given solution to a composition problem in polynomial time, we conclude that service composition problem is in NP.

By showing that the service composition problem is NP-hard and in NP, we have shown that it is also NP-complete. Note that the decision version of the service composition problem, which decides if there is a composition below a cost c_{req} is NP-complete; the optimization problem, which calculates the least cost composition, is NP-hard, similar to the set cover problem. $\square$

6.1.2 Approaches for Service Composition in Sensor Networks

The algorithms that we present in this section aim at achieving cost optimization across the sensor network, while reacting to changing network conditions by recomposing the service. The two proposed approaches are top-down (similar to *backward chaining*), that proceeds with composition down the service hierarchy, and bottom-up (similar to *forward chaining*) that proceeds in the opposite direction. To ensure that the heuristics terminate, we consider only service graphs that are *acyclic* and *directed*. We leave the formulation of service composition costs on more general graphs to future work.

6.1.2.1 Top-down Approach

The top-down algorithm starts when the user-request (which is represented by a *sink* service) is received. It first finds a set of services that satisfy its inputs and minimize the local cost, which is the sum of the cost of services chosen and communication costs between those services and user-request. Later, the chosen services choose their input providers and so on. A key requirement in this scheme is that the services at the same level should compose one after each other. It is easy to see that this approach is indeed a breadth-first traversal in the service graph, G_S, which provides us with the ability to identify which services are already used for composition. However, this breadth-first traversal requirement also makes the top-down approach difficult to implement in a distributed way, since it requires synchronization among sensor nodes.

At each level, a set of services is chosen such that among all sets that can supply the inputs to the service under consideration, the chosen set exhibits the smallest cost. To make such a choice, we use a well-known heuristic for the set cover problem, which chooses the service that adds the smallest cost per each input covered. *Critical services*, which are those that exclusively provide certain input fields of a service, are also selected. Since these have to be included in any feasible set of services, they are chosen first, in case they also cover additional inputs that would have to be provided otherwise by other non-critical services.

A drawback of the top-down approach is that, at any level, it may choose a set of input providers that cannot be further decomposed (their inputs cannot be satisfied), since it makes local decisions without knowledge of available services at the lower levels of the graph. Each service knows only its immediate neighbors in the service graph.

6.1.2.2 Bottom-up Approach

The bottom-up approach sorts topologically the directed and acyclic service graph G_S. This means that each service waits for its candidate input providers to decide on which services they will activate, before itself reaches a decision. Algorithm 3 presents a single level algorithm which composes the input that a service requires assuming that its neighboring services have already run the composition algorithm separately and know the set of services they would use for satisfying their respective inputs at a minimum cost. An important point to note in this algorithm is the presence of a $filter$ function that filters the neighbor list of the service under consideration according to the conditions set by the user. Metadata of the possible input providers of a service are considered, and only the neighboring providers that satisfy certain conditions on the metadata (e.g., service reliability, location of measurements, etc.) are included in the composition graph. Algorithm 3 uses a function called *find_comp* that selects the set of services with the smallest cost.

If the bottom-up method is implemented in a distributed manner, it incurs

Algorithm 3 Service Composition Algorithm with an Abstract Method for Choosing a Set of Services used for Input

```
method compose_service_inputs(S)
S.composition_cost = 0
$input_S$ = set of inputs of S
$N_S$ = filter(neighbor list of S,condition list)
for each neighbor $N_i$ in $N_S$ do
   $input_i$ = set of outputs of $N_i$
   $input_N[i] = input_S \cap input_i$
   $cost_N[i] = N_i$.composition_cost
end for
(S.chosen_services , S.composition_cost) = find_comp($input_S$,$input_N$,$cost_N$,S)
```

significant communication overhead to transfer complete composition subgraphs (the full list of services used by a possible input provider) among the neighbors of a service. The alternative approach of only transmitting the additive composition cost of the service upstream is followed instead. Once a service *S* chooses the set of its input providers, only the cost is transmitted further upstream to services that may utilize *S*'s output. Sending the cumulative cost information incurs a lighter traffic load on the system. However, less information is also made available about the composition of a service's possible input providers, which does not facilitate service reusability, and therefore may result in less cost-efficient compositions.

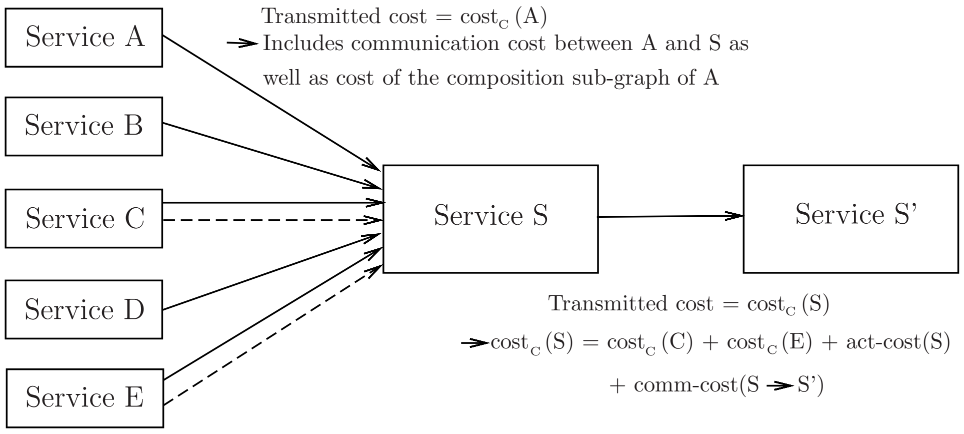

Service A
Service B
Service C
Service D
Service E
Service S
Service S'
Transmitted cost = cost_C (A)
→ Includes communication cost between A and S as well as cost of the composition sub-graph of A
Transmitted cost = cost_C (S)
→ cost_C (S) = cost_C (C) + cost_C (E) + act-cost(S) + comm-cost(S → S')

Figure 6.3: Sending the Collective Cost Information Upstream

As an example, in Figure 6.3, service *S* has five choices that can satisfy its inputs, and chooses two of them, *C* and *E* (double arrows) to utilize. *S* sends its cost information in its *metadata* to S', as the sum of the collective costs ($cost_C$) of the services it utilizes, its own activation cost (*act-cost*(S)) and its communication cost to S' (*comm-cost*$(S \to S')$). Such information transfer occurs throughout the service graph, G_S.

Algorithm 4 Algorithm for Choosing Services to Cover Input Set

```
method find_comp(input,output_sets,cost_set,S)
remaining_in = input
remaining_out = output_sets
chosen_services = $\emptyset$ , comp_cost = 0
while remaining_in != $\emptyset$ do
  if $\exists$ $S_j \in$ remaining_out is a critical service then
    $S_{min} = S_j$
  else
    for each service $S_j$ in remaining_out do
      Find $S_{min}$ where $\frac{cost\_set[S_{min}]+comm\_cost(S_{min} \to S)}{|remaining\_in \cap S_{min}|}$ is smallest
    end for
  end if
  chosen_services += $S_{min}$
  comp_cost += cost_set[$S_{min}$]+comm_cost($S_{min} \to S$)
  remaining_in -= remaining_in $\cap$ $S_{min}$
  remaining_out -= $S_{min}$
  if remaining_out == $\emptyset$ then
    break
  end if
end while
return (chosen_services , comp_cost)
```

The *find_comp* function required in Algorithm 3 is presented in Algorithm 4, and follows the heuristic for the set cover. At each step, it attempts to find the neighboring service that has the smallest cost per new input field that it can provide. The cost is calculated by adding the composition cost of the neighbor node (see Figure 6.3) and communication cost between the neighbor and the service that is being composed. As in the top-down approach, critical services are always included first due to the input fields that they exclusively provide.

In the evaluation section, we will show that the bottom-up approach achieves lower cost than the top-down approach, and, if there is a satisfying composition solution, it always finds it. The disadvantage of the bottom-up approach is that it requires an acyclic service graph G_S to terminate, hence the respective assumption.

6.1.2.3 Complexity and Approximation Ratio Analysis

The time complexity for the top-down and bottom-up algorithms can be computed as follows: for the top-down algorithm, each service looks at only its immediate input providers, or neighbors, and each service can have at most $|V|$ of them, where $|V|$ is the total number of services in the system. Since each service makes input checks as well as cost checks, $O(|V|^2)$ operations are performed at each step to choose a set of input providers using the greedy algorithm to minimize cost. Since only a fraction of services could be choosing the best service among the remaining ones, the top-down approach takes $O(|V|^3)$ time to complete.

In the bottom-up algorithm, for each service (Algorithm 3), there is a single call to $find\_comp$ (Algorithm 4) to find its respective set of input providers. This algorithm furthermore is just an implementation of the set cover heuristic, which takes $O(|V|^2)$ as in the case of top-down approach. Since $find\_comp$ is called once for each service, the complexity of the overall bottom-up algorithm is $|V|$ times the complexity of $find\_comp$, i.e. $O(|V|^3)$, which is the same as the top-down approach.

As it was previously shown, sensor service composition problem is NP-complete and we proposed two approaches that use heuristics. We will now give a brief discussion on their approximation ratio. Feige has proven that the lower bound for approximation of set cover problem is $(1 - o(1))$ lnn [142] unless there are quasi-polynomial algorithms for the problems in NP (n is the size of the set to be covered). This lower bound is achieved by the greedy search that chooses the subset that covers the uncovered elements of the set by choosing the smallest cost per element at each step. This algorithm has a $O(ln\text{n})$ approximation ratio and the method $find\_comp$ based on this greedy solution is given in Algorithm 4.

In conclusion, the greedy algorithm finds a composition of cost at most $log\text{n}.opt$ at each step of composition, for each service. In the formulation, opt is the cost of the optimal solution and n is the number of input fields that should be satisfied by a service. From this analysis, the overall composition that satisfies an end-user's request can be at most $log\text{n}^k.opt$, where k is the furthest distance from any source service in a composition graph to the final output formed by this composition. k can be at most the depth of the acyclic service graph G_S.

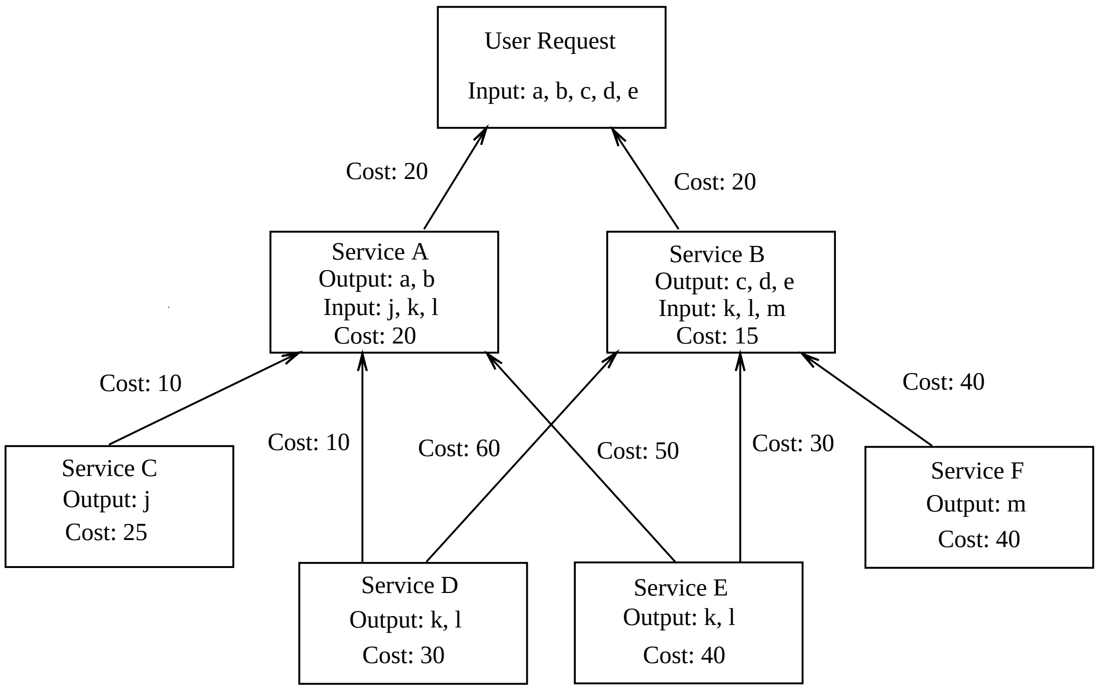
6.1.2.4 Implementing the Composition Decision Algorithm

The decision for composition can be made either in a *centralized way*, at a central decision node by receiving information from all services, or *distributedly*, wherein each service separately chooses (locally at the sensor node that they reside on) which services it will use to satisfy its inputs. The details of these two approaches are discussed next.

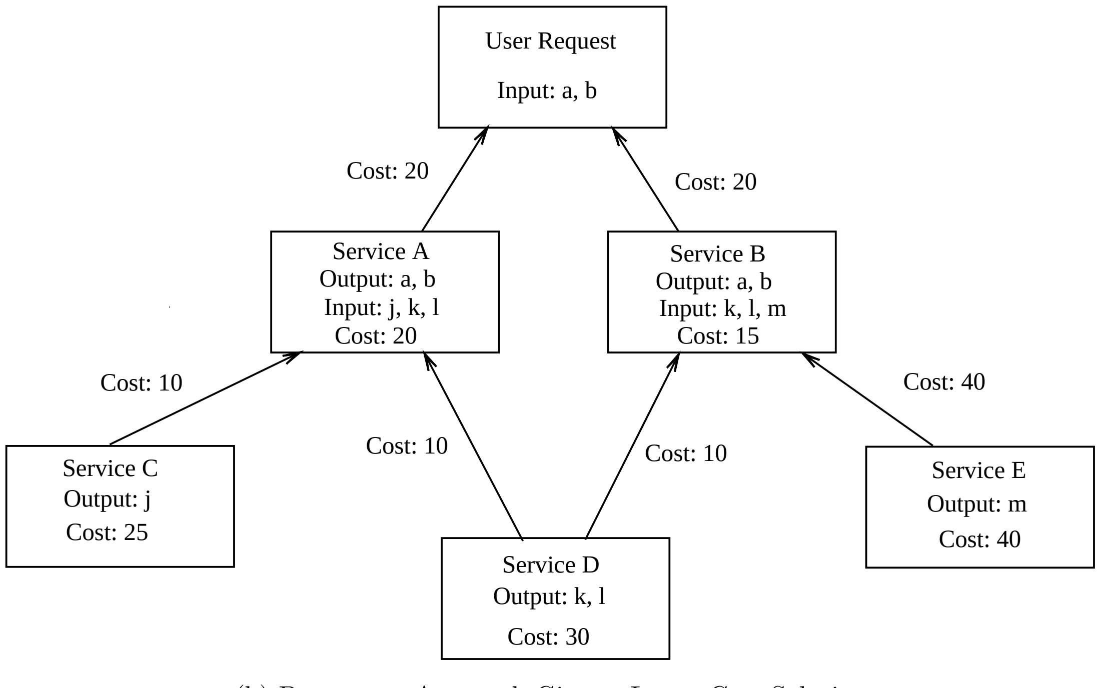
Centralized Implementation The centralized approach makes use of a central decision making node where metadata from each service is first collected. Once metadata is received from each service, the decision maker can run either the top-down or bottom-up algorithms and decide on the services to be activated. In our implementation, we use both algorithms and select the composition with lowest cost.

In the example of Figure 6.4, two sets of services are shown, along with their service graphs. In the first case (Figure 6.4a), the top-down algorithm will perform better. The service *user-request* will choose both A and B to satisfy its input. Then, A will choose C and D, and B will choose F. Since D is already chosen, B will also choose D to satisfy the input set $\{k, l\}$. The bottom-up approach however, would choose E for service B since it has lower cost, resulting in the utilization of two separate services for the same input fields ($\{k, l\}$). Note that B cannot check if D is already being used, since it cannot know whether the service that utilizes it will be chosen for the composition or not. For the second case of Figure 6.4b, the bottom-up approach will give a better solution since the top-down approach will choose service B, not knowing that, to satisfy B's input, service E has to be activated as well, which exhibits high cost.

Distributed Implementation In the distributed implementation, each service decides independently on which services it will activate in order to receive its inputs. Please note that what we mean by distributed composition is not to calculate the composition in a parallel manner among several nodes. This would still require the collection of all service information at one central node. Rather, each



(a) Top-down Approach Gives a Lower Cost Solution



(b) Bottom-up Approach Gives a Lower Cost Solution

Figure 6.4: Two Cases in which Top-down and Bottom-up Approaches are Better than the Alternative

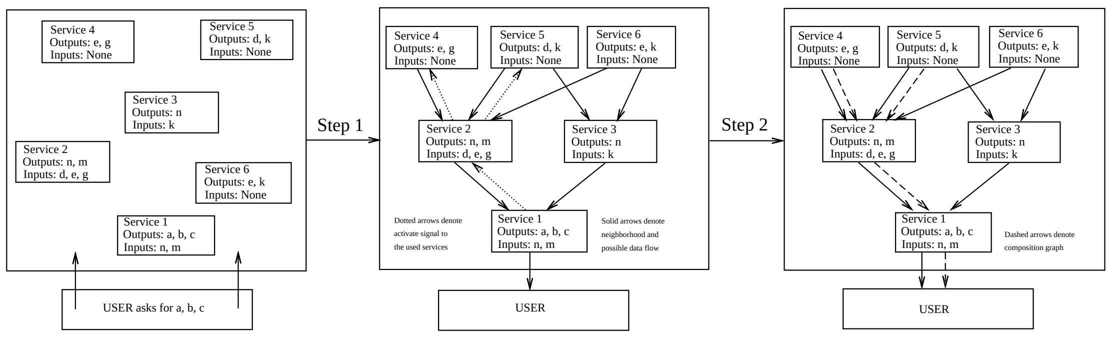


Figure 6.5: Service Composition Process in the Distributed Algorithm

service receives the input/output information (and other properties included in the metadata) of its potential input providers (sent to it again by these potential input providers), to choose a subset of them and hence satisfy its inputs locally, on the sensor node it resides on (hence removing the need for a centralized decision maker). The advantage of this scheme is its robustness to network faults and the quick reaction to a change in the network conditions. For a composition to change, services do not have to send their information to a centralized decision making point, which might constitute a single point of failure and bottleneck. The composition process is performed from bottom to top. In the distributed algorithm, a service will only have information about its own neighbors, i.e. the services which could provide input to itself. This information is all part of the metadata that the service received from its potential input providers, as described in Section 6.1.1.

When a composition process starts, messages are sent from the end-user requesting service (considered to be at the top level of the service graph) to lower level services. User-requested data has certain properties that are provided at the time of the request. According to these properties, a set of services whose outputs can satisfy the request will be considered neighbors of the user-request. This process repeats until this information is disseminated downstream to all the source services, which do not have any incoming edges, i.e. any inputs. Once this information has reached the source services, the reverse dissemination process takes place, in which, at each stage, the service composition algorithm is executed. Once every service is aware of its smallest composition cost, backward messaging takes place, where certain services are activated. This finalizes the composition graph. In Figure 6.5, a distributed composition example is presented. The solid arrows denote the possibility of information flow between services in the service graph G_S, for which the intersection of the input and output sets is not empty. The dotted arrows denote the activation decision of services after running the distributed composition algorithm. The dashed arrows denote the final composed graph, which is activated for operation. Since the composite service depends on primitive services whose execution might be affected by the dynamically changing network conditions, the composition graph can change during the operation lifecycle of a composite service. However, the service graph (denoted by solid arrows) will not change as long as the properties of the data that the user requests do not change.

6.1.2.5 Dynamic Composition

In a sensor network environment, the conditions of services can change fairly frequently. Hence it is important to be able to dynamically change service composition. For this purpose, we rely on *metadata* information exchange throughout the network.

For the centralized implementation, dynamic composition is very similar to producing the initial one: for each change in the system (e.g. change of a service's cost, or availability), the composition process is re-executed due to the new costs and available service set. The notification of the changes in the network conditions are sent to the centralized decision maker to make such recomposition possible.

In the distributed case, each service will decide on its new input providers based on the updates in its neighbors (i.e. candidate services to provide input to this service). When a service updates its own information (e.g. activation/deactivation, change of processing cost), it also notifies the other services that may utilize its output, so that they may change their respective compositions (i.e. which services they utilize to satisfy their inputs) if they choose to do so. Furthermore, if a service (or a link of a service) is not utilized anymore by the currently activated composition, a signal will be sent to this service to stop itself or one of its links. If the service is stopped altogether, due to not being used by any services currently in the active composition, this service also sends stop signal to every service it utilizes to form its input. This way the current composition (the subset of services in the network chosen to satisfy a user requirement) can be locally updated by the services that are in this composition. Of course such a distributed scheme can fall into local minima, as opposed to recomposition done from scratch in centralized implementation (we show this effect in our evaluations). Furthermore, if a service goes down (due to faults in the network) before it can send a signal to its neighbors, this could affect the dynamic composition. Such a problem can be solved by having services send periodic *active* signals to their neighbors, and once such a signal cannot be received for a certain time frame, the service can be assumed inactive.

Finally, an issue that we should take a close look at in dynamic composition is the frequency of the updates between services. Some of the metrics, such as energy left on the nodes on which the services are implemented, change constantly. Therefore a dynamic composition solution will be constantly at work, giving a high overhead. Such a situation can be prevented by setting up a composition update period, or an updating scheme. A service can furthermore wait for significant (e.g., threshold on reliability) changes in its condition before notifying the central decision maker or the neighboring services. These changes are application specific, so should be set by the user.

6.1.2.6 Mapping from Sensor Nodes to Service Graph

So far we have worked with services on an abstract level assuming that the services graph had already been given to us. The mapping of sensor nodes to service graph however is not trivial. Assume that there are k instances of a service A which can be instantiated on k sensors. Naturally, each of those instantiations correspond to a node in the service graph. Hence, two instances of the same service may have different communication costs if they are implemented on different nodes.

It is easy to create the service graph for the centralized implementation since each and every node will be sending its information to the centralized decision maker who can just generate the graph from the global information. The information sent by the sensor node is the set of its services (and their metadata) as well as its communication costs to its neighboring nodes. In fact, this information can be reduced just to a list of neighboring nodes since that lists all edges originating in the node and the cost is roughly the same for all neighbors assuming that similar amount of information is sent through each single hop. The neighbors are basically found by having each node broadcast a message once. The change of the topology can be detected by either a signal from the node whose state changes, or by lack of a signal if a continuous sequence of periodic *active* signals is organized.

It is harder to construct the service graph in the distributed case. Finding which services can send their information to which other services and at what costs requires every node to have global information of the entire network. Hence, there should be a distribution of service metadata between nodes. Once each service knows its neighbor services (which may be on different nodes, hence an edge in the service graph does not necessarily mean that the nodes on which those services are implemented are neighbors), the changes in the costs can be exchanged between the services since now the nodes know the route from one service to another. This way, distributed composition (bottom-up approach) can be accomplished. To disseminate service information of all nodes to all other nodes in the sensor network, a simple protocol like flooding can be used. Furthermore, such a scheme can also help to eliminate certain links between services. For example, if the node on which service A is implemented is more than α hops away from another service B, then B may not acknowledge A as a neighbor at all. By putting a *time to live* information into the packets searching for services, a virtual service graph in the sensor network with a time or hop limit can easily be constructed.

6.1.3 Evaluation of Initial Composition via Numerical Experiments

We have created 10000 cases to evaluate composition performance of our algorithms. There are 40 basic cases, each with unique number of services varying from 1 to 40 services. We further created 250 variants of each basic case. In each variant, we assigned each service a uniformly distributed cost between 0 and 40 and communication cost between services uniformly distributed between 0 and 50. We first create the acyclic graph and then assign inputs and outputs according to the created edges. We also choose a random user-request from the set of inputs and

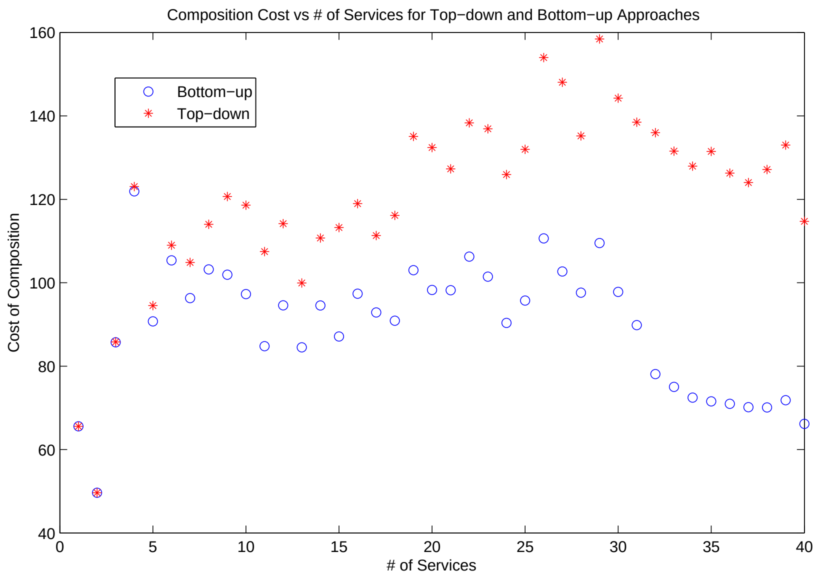


Figure 6.6: Cost of Composition Comparison of Top-down and Bottom-up Approaches

outputs defined in the graph. Later, we run our algorithms on this setting and collect the performance results. Please note that this section provides the numeric evaluation of how the top-down and bottom-up approaches perform in creating a low cost composition, hence the cases are more likely theoretic than belonging to real world applications.

Results: In Figure 6.6, it can be seen that the bottom-up approach constructs lower cost compositions for the given cases in average, compared to the top-down approach. Such an evaluation can be misleading however, leading one to use always bottom-up approach. In the next figure (Fig. 6.7), we show that the approach of using the best of top-down and bottom-up in the centralized implementation is a clever choice, and gives lower cost compositions. This is due to the fact that although the bottom-up gives much better results than top-down in average, the best of both gives better results since top-down approach performs better in certain cases. We provided one example of such cases in Figure 6.4a.

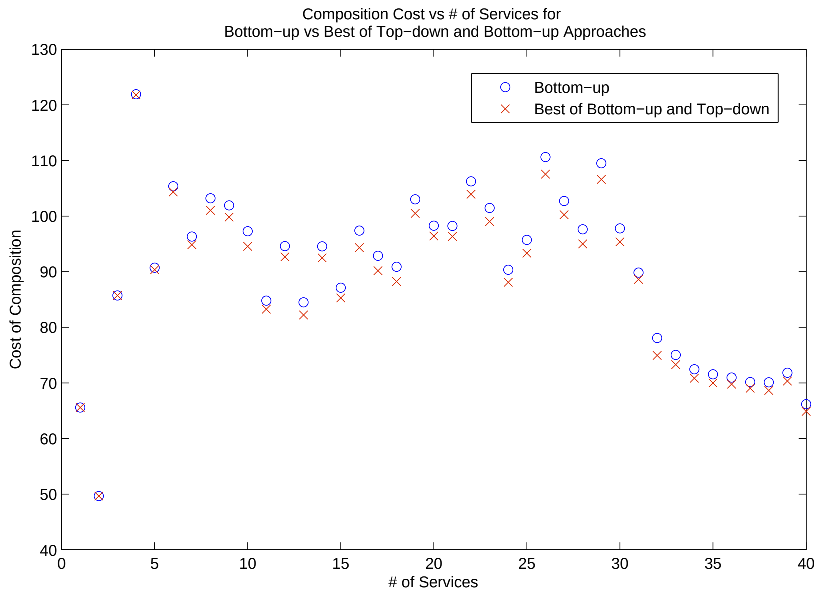


Figure 6.7: Cost of Composition Comparison of Best of Bottom-up and Top-down vs Bottom-up Approach

6.1.4 Evaluation of Dynamic Composition in ns-2

To evaluate the dynamic composition capabilities of the distributed and centralized schemes under changing network conditions and node faults, we implemented a composite sensor service application scenario in the ns-2 simulator. The centralized implementation uses the best of bottom-up and top-down approaches while the distributed implementation uses only the bottom-up approach, which sends only the collective cost upstream. We present the overhead of both approaches, the cost of the compositions they provide over the simulation period, the activation ratio for a feasible composite service and the time it takes for both approaches to react to changing network conditions.

Application Scenario: Figure 6.8 illustrates the application scenario that serves as the basis for the simulations. It assumes a monitored field consisting of six areas labeled A to F. In each area, there are three types of source services: low, medium and high quality visual monitoring (e.g. *A High Quality*). There are also intermediate fusion services that receive input from two area services and provide an

The top-down approach implemented in a distributed manner requires synchronization among nodes.

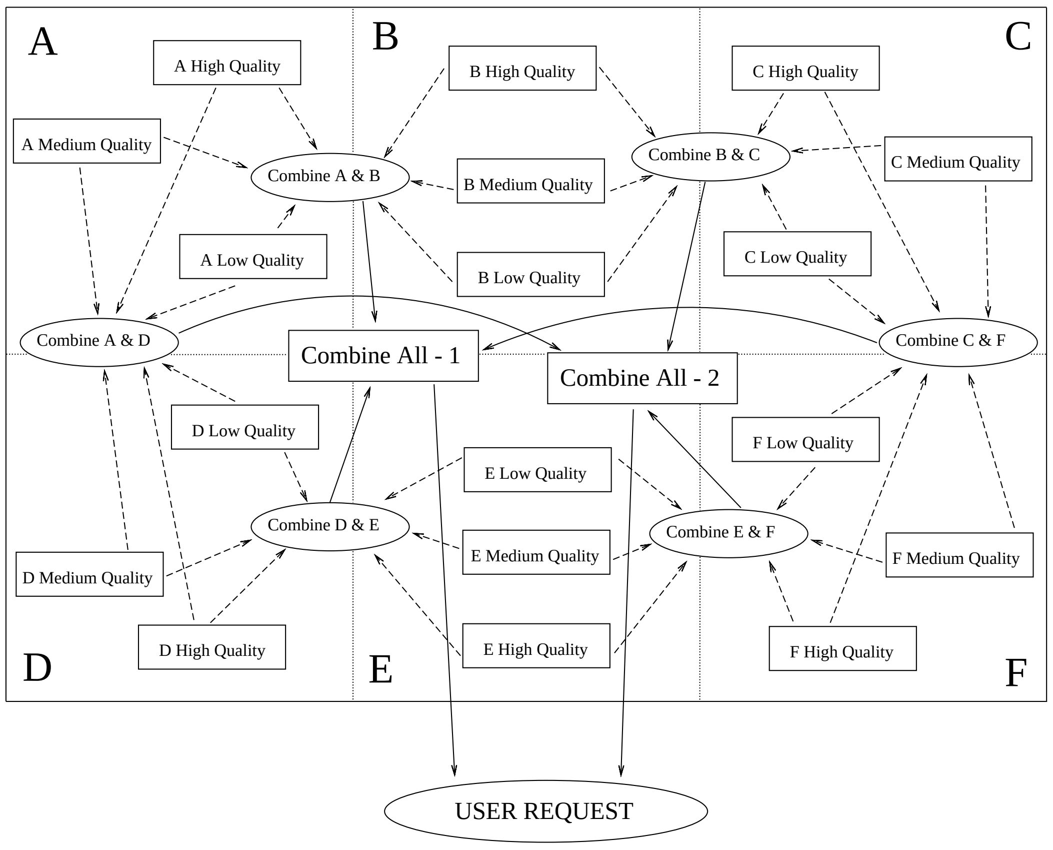


Figure 6.8: Simulated Sensor Network Application

intermediate result whose exact functionality is not important for the purposes of the simulation (e.g. *Combine A & B*). It is assumed that these intermediate services also come in three variants (e.g. *Combine A & B High Quality*), not all of which are shown in the figure, for simplicity. There are also several final fusion services that use the output of the three intermediate services to provide a result for the whole area, e.g. *Combine All - 1*. The user request is submitted through one of the fusion services.

The processing and communication costs have been set according to the size of the outputs produced. For example, the processing cost of the source services were set as 20, 30 and 40 cost units for *Low*, *Medium* and *High Quality* respectively, following the size of the outputs of these services that was set to 4, 8 and 12 size units. We also assume that the *communication cost* per hop and size unit is 5. For example, transmitting the output of a high quality source service between two sensor nodes over a single hop would cost $5 \times 12 = 60$ cost units. The intermediate fusion services that process *High*, *Medium* and *Low Quality* have processing costs of 20, 30 and 40 cost units respectively. The size of their output is set to 6 size units. Lastly, each fusion service has a processing cost of 15 cost units, and an output size of 3.

Implementation in ns-2: The proposed service composition schemes are implemented in ns-2 as an application-layer protocol, independent of the exact underlying routing and MAC layers used for the messaging among nodes. For our simulations, we used *AODV* and *802.11 MAC* as provided by the ns-2 distribution in version 2.34.

Our ns-2 implementation consists of two application layer agents: *ServMeta* and *ServHolder*. ServMeta agent represents a service description, and includes the properties of the service (i.e. the service metadata). A new instantiation of a ServMeta agent is defined for each distinct service (i.e. each service type has a ServMeta agent in the ns-2 script file). ServHolder agent on the other hand is assigned to each and every node on the network, and is a container agent which the ServMeta agents belong to. ServHolder agents message between each other to transmit the information of the services assigned to themselves (i.e. the services that are assigned to the nodes they reside on).

Services are assigned to nodes as follows: each source service is assigned randomly to one of the nodes in its relevant area of monitoring, and there is one instance for each level of quality. Two instances of each intermediate and final fusion service are activated and a single instance of user request is assigned to a sensor node, which never gets deactivated throughout the simulation time. Additionally, for the centralized implementation, we assign a decision maker to one of the nodes that gets deactivated just like the other nodes, to quantify the effect of single-point-of-failure.

After service assignment, the composition protocol for both the centralized and distributed approaches starts with the *service discovery phase*: each node floods metadata (cost, type and size of outputs/inputs) of the services that reside on it to every other node. This way each service discovers what other services exist in the network, and if they are neighbors in the services graph (i.e. can provide input to or receive output from them). It also learns the hop-distance between nodes, which is needed to calculate communication cost between services. Messaging cost for this discovery stage is $O(n|S|)$ where n is the number of nodes in and $|S|$ is the number of services that reside on the sensor network (since the messaging is between the ServHolder agents of the nodes that contain the descriptions of these services and not between the services, i.e. ServMeta agents, themselves).

In the centralized composition approach, following the service discovery phase, the centralized decision making node broadcasts a request for service metadata information including hop-distance between every node in the network. Based on the global knowledge that is collected, it then composes the service graph that satisfies the user request and sends a notification to all services that have been selected during the composition phase. The process is repeated whenever a service changes its activation status or processing cost, which is made known by the node that it runs on by sending appropriate status notification messages to the centralized decision making point. The composition process can also be delayed if the node on which the composition process runs (centralized decision maker) becomes inactive (i.e. single-point-of-failure).

The distributed composition scheme starts with the service that is assigned with the user request sending compose-request messages to its neighbors, after the service discovery phase. These neighbors in turn propagate this request to their neighbors and this process repeats itself. A service composes itself (i.e. chooses its input providers) once it receives notifications from all its neighbors that they themselves have completed their composition (chosen a subset of their own possible input providers to satisfy their requirements) with their additive cost (Figure 6.3), at which point it proceeds to choose its input providers. Once the service with the user request composes itself, a backward messaging procedure takes place, as described in the example of Figure 6.5 to notify the services that have been selected. In case of change of activation status of a node, a notification message is sent to the immediate neighbors of the services that run on that node to perform a local recomposition (similarly for a service that has its processing cost changed). If the local recomposition is not feasible, then the request is propagated upstream in the

Table 6.1: NS-2 Simulation Parameters

Parameter	Value
Field Size	120x80
Area Size	40x40
# of Areas in Field	6
Comm. Radius	50
Node Activation Ratio	60%–100%
Node Inactivity Time	160–240 secs (uniformally distributed)
Service Processing Cost Change Frequency	Every 20–400 secs
Service Processing Cost Change Period	1600–2400 secs (uniformally distributed)
Service Processing Cost Change Percentage	+ 0%–40 % (uniformally distributed)
# of Nodes	30 (5 per area)
Total Simulation Time	20,000 secs

service composition graph, until a valid composition is achieved.

Simulation Setup: The parameters of the simulation experiments run in ns-2 for the aforementioned application scenario are shown in Table 6.1. *Node Activation Ratio* is used to calculate the expected amount of time a node remains inactive (with the services that run on it being unavailable) compared to the expected time it stays active for the duration of the simulation, which is $\frac{1-activation\_ratio}{activation\_ratio}$. *Node Inactivity Time* denotes the amount of time a node remains inactive, once it changes its status to being so. We also present the results of an experiment where the activation ratio was kept at 100% and we looked at the effect of cost changes for services in the network. *Service Processing Cost Change Frequency* represents the inter-arrival time of events when we increase the cost of a random service in the network by 0%-40% (*Service Processing Cost Change Percentage*). The service goes back to its original cost in 1600-2400 seconds (*Service Processing Cost Change Period*).

Results: Our first experiment, the results of which are given in Figures 6.9, 6.10 and 6.11, looks at the effect of activation ratio of each node in the sensor network during the simulation period (the processing costs for services are kept same for these experiments to see the effect of node faults) as given in Table 6.1. These experiments evaluate the usefulness of centralized and distributed composition scheme implementations in the erroneous conditions of the sensor network.

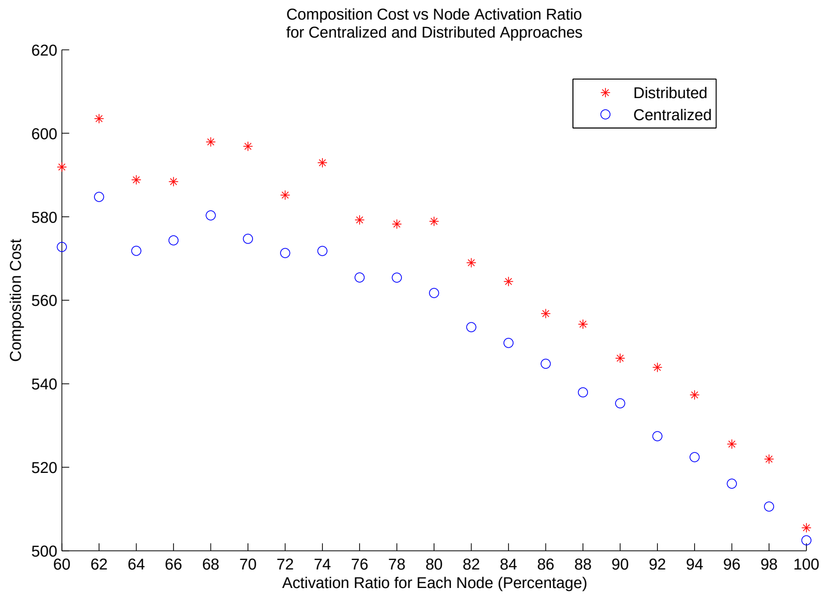


Figure 6.9: Comparison of Composition Cost for Centralized and Distributed Approaches with Varying Node Activation Ratios

In Figure 6.9, the composition costs of the centralized and distributed approach as a function of the activation ratio of each node in the sensor network during the simulation period is shown. Each value represents the average of 10 runs for each activation ratio, and the cost is calculated only for those times during the simulation when a feasible composite service existed. It can be seen that, overall, the centralized approach performs better than the distributed approach as it generates composition graphs that exhibit lower total processing and communication cost. This is due to two factors: first, the centralized implementation chooses the best of top-down and bottom-up approaches. Second, the recomposition process of the distributed scheme is performed locally, hence it can fall into local optima during the simulation. Centralized composition recomposes the service each time a change in the status of a node takes place. From the figure, it is also clear that there is a decreasing trend for both approaches in terms of composition cost as the node activation ratio increases, since the more service instances become active concurrently, the higher the probability that a solution is found with a lower cost that would require fewer recompositions. It should also be noted that the centralized

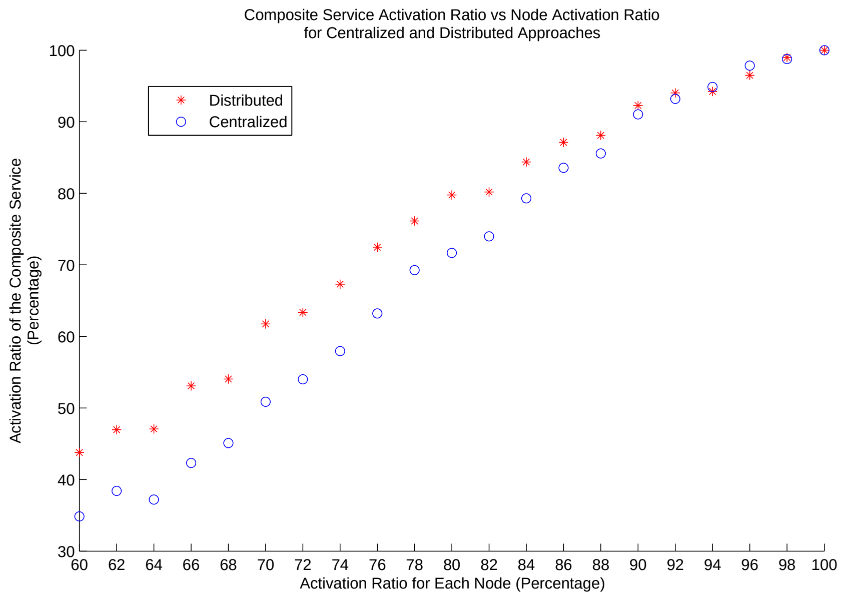


Figure 6.10: Service Activation Ratio Comparison of Centralized and Distributed Approaches for Varying Node Activation Ratios

implementations applied by previous work (such as [126]) can perform with coming up as efficient compositions as our centralized scheme does, provided the similar algorithms are applied. Our one advantage even for the centralized implementation is that we choose the best of top-down and bottom-up approaches.

Figure 6.10 shows the service activation ratio, defined as the percentage of simulation time when there was a feasible, active composite service, of the centralized and distributed approaches. It can be seen that due to the local recomposition process, the distributed implementation is more resilient than the centralized approach, due to its higher service activation ratio. Furthermore, it is more robust to the single-point-of-failure of the decision making node that plagues the centralized approach, which becomes deactivated once such a failure occurs. Please note that any composition scheme, including the logical programming based approaches described in the related work section, can perform no better than the centralized version we apply in terms of activation ratio. The distributed decision making we propose in this section brings higher robustness in a highly erroneous environment such as a sensor network, hence is an improvement over previous work.

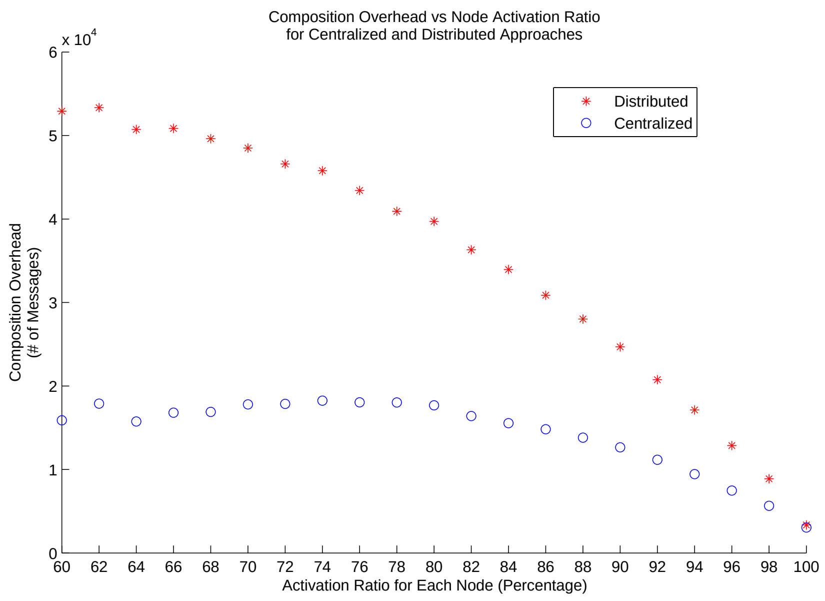


Figure 6.11: Overhead Comparison of Centralized and Distributed Approaches for Varying Node Activation Ratios

The respective comparison regarding the messaging overhead, measured as the number of messages that are exchanged between two nodes is shown in Figure 6.11. The plot also includes the overhead caused by messaging during the *flooding phase*. Clearly, the distributed scheme exhibits much higher overhead than the centralized one because changes of a service have to be broadcasted to all of its neighbors as opposed to a single centralized decision making node. Note that there exists a certain trade-off between the activation ratio versus the composition cost and overhead between centralized and distributed approaches, making the choice between these two schemes application-specific. For example, an application that can afford low activation ratios but has to be energy-efficient may run the centralized composition scheme, while a mission critical service that requires to quickly recompose upon a node/service failure would be more efficiently composed under the distributed implementation.

Our second experiment keeps the node activation ratio at 100% to solely look at the effect of processing cost changes for services in the sensor network. As aforementioned, in this section, we use an abstract notion of cost (processing and communication), which can be interpreted as both quality of service metrics, or actual

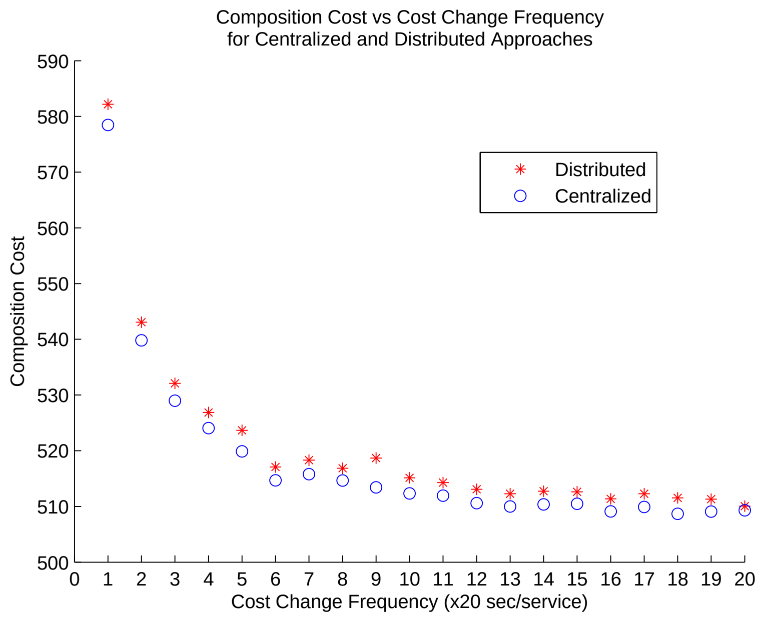


Figure 6.12: Comparison of Composition Cost for Centralized and Distributed Approaches with Varying Cost Change Frequencies

costs, such as energy spending. The results of this experiment are given in Figure 6.12 and Table 6.2. These experiments compare the centralized and distributed composition scheme implementations in the face of changes in the sensor network conditions.

In Figure 6.12, we compare the distributed and the centralized composition implementations in terms of the cost of the compositions constructed, where the frequency of service cost changes varies between every 20-400 seconds in steps of 20 seconds. For example, if the x-axis shows 12 (i.e. 12x20 = 240 secs), this means that every 240 secs (actually every 216-264 secs with uniform distribution, with mean of 240 secs), a random service in the sensor network is chosen to increase its processing cost by 0%-40% (uniform). This service goes back to its original cost in 1600-2400 secs (uniform). For each frequency level, we ran 10 cases, each running for 20000 seconds. As expected, Figure 6.12 shows that increase frequency brings higher cost compositions for both approaches. Centralized implementation again performs slightly better, and the reasons for this are similar to the aforementioned experiment with changing activation ratios.

In Table 6.2, we show the average reaction times of both approaches to a

Table 6.2: Reaction Time Comparison of Centralized and Distributed Approaches for a Service Processing Cost Change in the Sensor Network

	Distributed	Centralized
First Reaction (sec)	0.0751	0.0080
Last Reaction (sec)	0.3198	0.8036

change of processing cost in the sensor network. The first reaction time represents the time it takes for an initial reaction (recomposition, adding or removal of a service to/from the current composition etc.) to a service's cost change in the system. We see that the centralized implementation performs better for this metric. The reasons for this behavior can be listed as follows. When a service changes its processing cost, there is only a single message sent to the centralized decision maker in the centralized implementation, while in the distributed implementation, each service that may utilize this service's (the one whose processing cost is changed) outputs receives a message. This leads to more initial messaging and the high load (and possibly congestion) leads to later arrival of messages to services which react to this processing cost change. However, we see that for the metric of last reaction, the distributed scheme performs better. To be exact, the last reaction represents the time when the last action is processed by the network (again, such as recomposition, adding or removal of a service to/from the current composition etc.), hence indicates the time when the composition is stable again. The lower last reaction time given by the distributed implementation shows that it comes up with faster recompositions, due to the local recomposition (whereas the alternative decides at the centralized node, and notifies the services that belong to the new composition scheme), as aforementioned. For erroneous and unstable environments such as sensor networks, this is the metric that matters, and the distributed implementation performs better, as expected. This result is also an indicator of distributed composition's robustness when the nodes deactivate, by showcasing its capabilities to react quickly to changes in the network conditions.

6.1.5 Related Work

Service composition has attracted constant interest in web services domain, where there are no constraints on resources and there is high user interactivity. The latter helps with semi-automated and manual composition approaches, reducing the need for automated composition techniques. Although there are detailed surveys on web services and composition [135], [143] as well as sensor networks programming [144], [145], below we discuss some papers that are more closely related to our work.

In the domain of web services, many works have focused on the subject of *QoS-aware web services composition* [122], [123], [127]-[129], where an abstract service composition graph is provided to the system, and the problem is to bind actual services to the abstract services on this graph. If the system tries to optimize an aggregate QoS function [146], comprised of different metrics, then this problem turns out to be a variation of the multi-variable assignment problem, hence it is NP-hard.

Of the references given above, Mao et al. [129] proposes *Automatic Path Creation* service (APC), which is a centralized dynamic web service composition method that considers quality of service (QoS) and network characteristics. This method looks for a shortest path from the end-user to the primitive services, but does not consider the case where a subset of the outputs of a service can be used as input to another service.

One of the main methods utilized in the literature to optimize the metrics considered in web service composition is genetic algorithms. In [122], the authors solve the problem of assigning (binding) concrete services to the abstract services in the already given service composition graph according to QoS measures via using the genetic programming approach. When the QoS values deviate, their method applies a recomposition, i.e. rebinding of the services. The authors of [127] utilize the functional metric, semantic quality fit of services, as well as QoS optimization (which is mainly a mixture of non-functional metrics) in assigning services to tasks in a composition. Again, a genetic algorithm based solution is applied to optimize the combined metric.

Another method applied in *QoS-aware web services composition* is the integer programming approach. In [123], integer programming is utilized in assigning concrete services to the abstract services provided in the task graph. Furthermore, the authors provide negotiation techniques to reach a feasible solution when the constraints set by the user are severe. Finally, a mixed solution to the problem of assigning services to the task graph is given in [128], where both local (i.e. where assignments to tasks are considered separately) and global (i.e. where assignments are considered together) optimization is considered to maximize QoS in web services composition. Please note that the assignment is polynomial time solvable for local optimization for *QoS-aware web services composition* (this is NP-complete in our case). For global optimization of QoS metrics, [128] utilizes integer programming techniques.

The problem we focus on in this section is inherently different from *QoS-aware web services composition* in many aspects. First of all, we do not bind services to tasks whose connections are already given in the pre-provided task graph (i.e. service composition graph). In our problem, the user presents a set of outputs that should be provided by the overall system, and the composition graph is generated due to the services at hand. It is not always possible to replace services (since the same information can be provided by different services on the sensor network, and the rest of the output provided by these services can be different), and as we proved, even the local optimization of cost is NP-complete. Our second difference lies in the composition mechanism. Previous work in *QoS-aware web services composition* provides centralized decision making, which can lead to frequent single-point failures in the domain of sensor networks. We provide a distributed decision making scheme, where each service in the sensor network decides its input providers locally. We show in the evaluations section (Sec. 6.1.4) that such distributed making leads to faster recomposition of services and therefore increases activation ratio of the composite service, hence increases robustness, which is a vital component in faulty environments such as sensor networks.

The works that are closest to ours apply logical programming, where the services are described as a set of pre- and post-conditions [124], [126], [147]-[149]. Furthermore, the methods of forward and backward chaining (similar to our bottom-up and top-down methods, relatively) are applied to satisfy the conditions set by the user.

Of the above papers, [124] proposes *OWLS-Xplan* which is a logical planner that utilizes the service definitions given in OWL-S (Web Ontology Language - Services) language. This way the conditions can be set by the user and satisfied according to the pre- and post-conditions of the services. Another logical programming solution is provided in [126]. This is a similar work to ours in that it utilizes forward chaining (similar to our bottom-up approach), and uses an aggregated QoS measure to choose provider services at each point of composition, and utilizes pre-/post-conditions, both on the properties and the types of inputs/outputs that match between services. Significant differences from our approach can be listed as follows: (*i*) we allow for distributed decision making (i.e. each service chooses its input providers locally on the node they reside on), which is crucial for sensor networks, (*ii*) we apply a mixture of top-down (i.e. backward chaining) and bottom-up (i.e. forward chaining) algorithms, which produces better results in the centralized composition scheme, (*iii*) we show the NP-completeness of even the local optimization problem (which is valid for this domain as well) and provide an approximate solution, and (*iv*) we address the communication costs and activation ratios due to services activating or deactivating, which is a crucial factor in sensor networks. These differences also apply to other logical programming based approaches.

Another example which is given in [147] provides a logical programming based composition of services based on pre-/post-conditions of services, however, it does not take into account the QoS or cost measures. In [148], MARIO (Mashup Automation with Runtime Orchestration and Invocation) is introduced, which utilizes tags chosen by the user to provide possible composition schemes. Each tag represents a functional goal and can be interpreted as a query. Type hierarchies are used to connect outputs of a service to compatible inputs of another service in the composition decision process. This work however, does not take into account the changes in the network for performing a re-composition.

Use of the OWL-S language to describe the web services with their inputs and outputs is discussed in [149]-[151]. The first paper introduces the methods for translating services described in OWL-S to SHOP2 compatible descriptions (a logical planner) to compose a user-request. The planner makes use of pre-/post-conditions of described services. The paper however does not take into account any type of distributed decision making, and also does not consider the maximization of QoS as some previous work did, or minimization of cost, as we do. Furthermore, in [150] and [151], the presented methods are user supervised in the sense that, at each step, a set of possible matches for the user functionality needs are presented to the user who selects one or more services for use.

In mobile networks, the authors of [152] propose the use of service (both semantical and syntactical) equivalence in order to replace services where the connections between nodes are changing rapidly. [152] again assumes a pre-provided composition graph for the initial operation, hence the replacement occurs with other services that can function as well as the current services in the graph. Furthermore, other approaches for service composition, such as Petri-Nets [153] (this work does not take into account any cost or QoS measures), have also been applied in literature.

In sensor networks, few approaches have been proposed for service composition. A significant one is [134] in which the authors provide a method based on logical programming through backward chaining for combining services. They model services as statements whose truth depends on their predicates and they set certain statements true when these predicates are satisfied. These statements are further used by other services as predicates. The method is used for automated inference in sensor networks. Another paper in the sensor networks domain [154] tries to identify the service composition that is less likely to be invalid in the near future due to nodes going to sleep mode etc. The goal is to minimize the recomposition cost at a later time. In [155], the authors propose components for a network of sensors and actuators from which the complex desired services can be composed. However, the composition process is entirely user-driven.

In [156], the authors propose a dynamic flow control solution, applicable to sensor networks, which uses filters and wires between services. By using filters on the wires (which are logical conditions), the user manually blocks data flow whenever such blocking is needed for the functionality desired in the current network conditions. Their system still requires user interaction. Another programming framework, *EnviroSuite* [157], abstracts external environmental elements into objects hence simplifies sensor network implementations. EnviroSuite is appropriate for implementing the service modeling we propose in this section, however it does not include automated composition, which to our knowledge is novel in sensor networks domain.

A model similar to our service graph can be found in [158], which proposes abstract task graphs that consist of abstract tasks and abstract channels. These are mapped to services (nodes) and possible connections (edges) in the service graph, respectively. However, this paper does not address automatic composition construction or cost measures. A final work worth mentioning is that of [159]. The authors present MiLAN, which is a middleware for sensor applications. MiLAN receives the application requirements in terms of the information needed and chooses a set of sensors that can provide this information according to certain quality of service requirements. However, MiLAN does not provide composition of services in which outputs of services are combined to provide inputs of other services.

6.1.6 Conclusions and Future Work

In this section, we have described a novel method of service modeling and dynamic composition which is appropriate for the unreliable and unstable sensor network environment. We have introduced two algorithms for composition of sensor services. These can be classified according to the direction that they follow on the service graph during composition. We also presented centralized and distributed approaches and their advantages and disadvantages, as shown by the dynamic composition evaluations. We furthermore discussed implementation details of the service model and the composition algorithm.

Our future work includes the realization of the algorithms on an actual sensor network application. Furthermore, we are planning to follow a game theoretic approach to choosing services with minimal cost. In such a scheme, each sensor node will aim to lower its load by utilizing services implemented on other sensors rather than itself. We believe that such a scheme will lead to lower cost solutions, after the initial composition is done with our current algorithms.

6.2 Utilizing PCFGs for Modeling and Learning Service Compositions in Sensor Networks

In the previous section of this chapter, we proposed a methodology on representing a composition as a graph of connection of services and how to automatically combine services in an efficient way. While such an automated scheme improves energy efficiency of the solution, during composition creation this scheme suffers from communication overhead and a security problem because metadata about services are exchanged between all potential services (to be included in a composition) in the network. Since the metada may include mission critical information, in domains such as military applications, the user may not want such information to be moved around freely. A possible solution, which has also been followed in the literature, is the manual connection of services by the user. However, such a method is error-prone to the degree increasing with the composition size.

The current section addresses the above problems of security, high overhead and performance involved with service composition in sensor networks. To be able to create high performance service compositions with low overhead, we propose to use probabilistic context free grammars (PCFGs) to learn the composition schemes in a sensor network. We assume that the efficient initial compositions of services are available, for example by creating compositions, evaluating their performance and collecting the most efficient one. They are used to create a PCFG such that each initial efficient composition for the given network is a sentence belonging to the language defined by this grammar. We also provide a method on how to generate a string representing a service composition. Each PCFG constructed via our method is specific to a single sensor network instance with predefined set of elementary services and possible information flows between them. In such a PCFG, a production rule with high probability assigned to it represents a highly efficient subcomposition step. The hidden assumption here is that the frequently used service combinations appear in the intial set of efficient compositions for a reason, because they are more advantageous than the alternatives. Hence, the constructed PFCG defines a *language* for efficient service compositions. In a future work we will consider creating PCFGs from the poorly performing compositions to learn also inefficient service combinations to avoid them.

The contributions of the current section can be summarized as follows:

- A PCFG and its corresponding language to represent efficient service compositions as strings in this language,
- A methodology for generating a PCFG from previous compositions,
- A method for finding frequently used subcompositions in PCFG sentences, which in turn can be represented as synthetic services, improving efficiency of the service design,
- A method to utilize the PCFGs for creating efficient compositions and reducing the commmunication and computational overhead of this process.

The rest of the section is organized as follows. We first explain the language for describing a service composition as a string, which basically is a way to represent a service composition graph. Then, in Section 6.2.2, we discuss the basics of our methodology. We provide a way of finding subcompositions while constructing a PCFG. We also explain why generating PCFGs is a better solution than just statistically inferring the connection probabilities, or simply examining one level service usage sequences. We also discuss how a composition is created from the constructed PCFG as the system runs. In Section 6.2.3, we compare the PCFG based composition methodology to our previously presented automated composition technique [119] (see Section 6.1) in terms of performance, as well as the previously examined modeling/learning techniques for this application. Next, we talk about previous work on service composition learning techniques in Section 6.2.4. We finalize the section with our conclusions and outline of future work in Section 6.2.5.

6.2.1 Describing Service Compositions as Strings

As aforementioned, we define service composition as a data flow graph where the directed edges represent the input being provided from the service at the start of the edge to the service at the end of the edge. Each service is represented as a vertice in the composition graph. To describe a composition graph as a string, we place service names within the parentheses to represent the services used as input providers. Hence, the following grammar for the language to desribe service composition graphs can be used, assuming that there are n elementary services defined in the given sensor network:

<composition> → <service_name>(<composition_list>) | <service_name>

<composition_list> → <composition> | <composition>, <composition_list>

<service_name> → service$_1$ | service$_2$ | ... | service$_n$

Hence, each node in the data flow graph is represented in the string by its name followed by the list, enclosed within parentheses, of nodes in the subgraph rooted in it. In the above graph grammar, we have just four tokens: *left and right parantheses* which encapsulate the set of services used by a service whose name preceeds the matching parantheses, *commas* to separate compositions, and *service names*. Such a grammar of course can only represent service compositions which are acyclic (i.e. the composition string is finite).

To give an example, we use the application from [160] which requires the composite *Camera* service. In Figure 6.13, the services $LOBR$ (with indices to distinguish them) give a measurement ($LOBR$ stands for Line of Bearing Report) of the acoustic sensor's distance from a source of sound regarded as an event. Later, these measurements from three acoustic sensors are sent to the $LOBR2LOCR$ service which uses triangulation to detect the location of the event ($LOCR$ stands for Location Report), and sends this information to the camera sensor (another service), which then starts monitoring the event. The $LOBR$ services require no input, so they use the "<composition> → <service_name>" rule of the language given above for their composition. However, $LOBR2LOCR$ uses services $LOBR_1$, $LOBR_2$ and $LOBR_3$, hence its composition is like a function call, which starts with the service's own name ($LOBR2LOCR$) and is followed by the set of services (with their corresponding composition strings) that are used to satisfy its inputs. Finally, the service *Camera* uses composite $LOBR2LOCR$ service, so its composition string contains both $LOBR2LOCR$ and its composition subgraph.

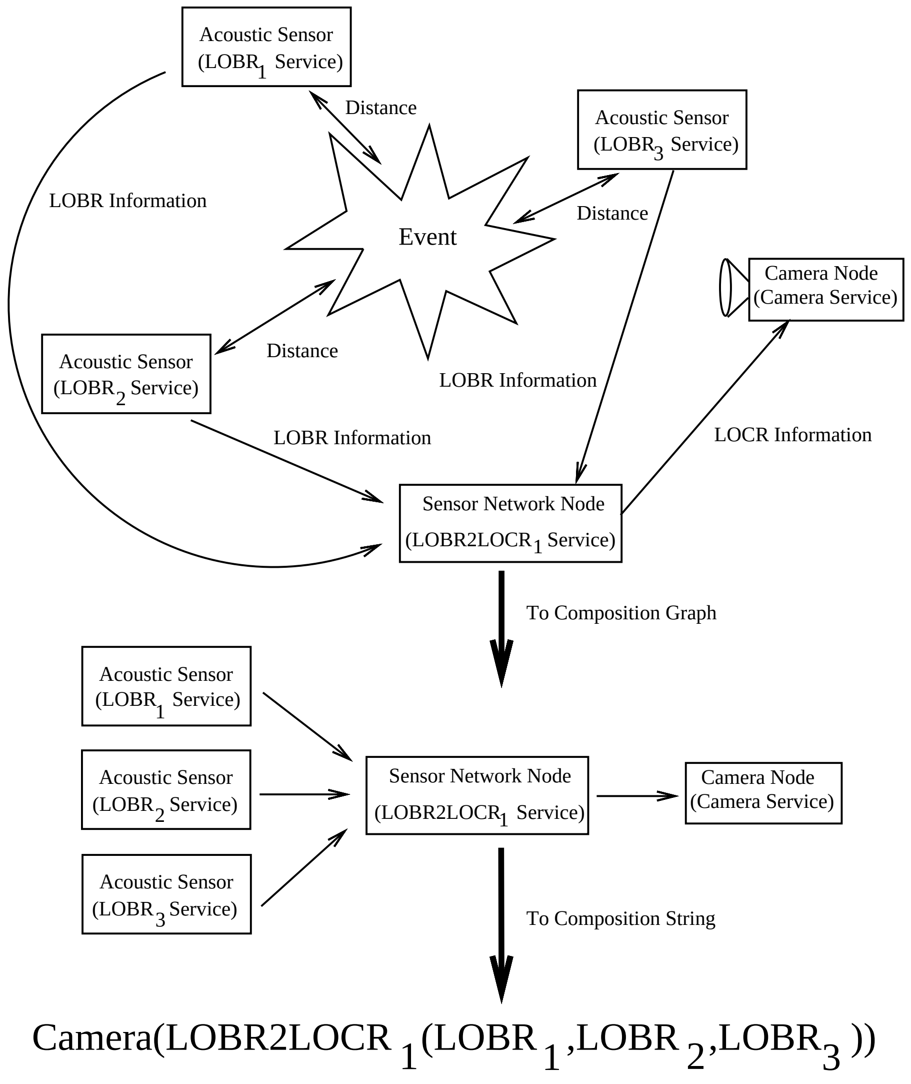

Acoustic Sensor (LOBR1 Service)
Distance
Acoustic Sensor (LOBR3 Service)
LOBR Information
Event
Distance
Camera Node (Camera Service)
Acoustic Sensor (LOBR2 Service)
Distance
LOBR Information
LOBR Information
LOCR Information
Sensor Network Node (LOBR2LOCR1 Service)
To Composition Graph
Acoustic Sensor (LOBR1 Service)
Acoustic Sensor (LOBR2 Service)
Sensor Network Node (LOBR2LOCR1 Service)
Camera Node (Camera Service)
Acoustic Sensor (LOBR3 Service)
To Composition String
Camera(LOBR2LOCR1(LOBR1,LOBR2,LOBR3))

Figure 6.13: An Example to Illustrate the Service Composition Language

A simple parser can extract the links from the given composition. Furthermore, as mentioned in the introduction, our methodology uses the strings of the previously run compositions to encompass in the PCFG the rules of combining services efficiently. Thus, the constructed grammar can be used to either create efficient compositions automatically, or to check the validity of the compositions for correctness and efficiency.

Let's provide a simple PCFG based on the example from Figure 6.13. For disambiguation, we are representing nonterminals with uppercase letters and terminals with lowercase letters. For our purposes, let's assume the camera service can be composed in two ways: "camera (lobr2locr_1 (lobr_1 , lobr_2 , lobr_3))" with 0.8 probability, and "camera (lobr2locr_1 (lobr_4 , lobr_5 , lobr_6))" with 0.2 probability. Then a PCFG corresponding to such a case can be provided as below:

$$\text{Start} \rightarrow \text{camera} \quad (\ \text{LOBR2LOCR}\ ) \quad (1.0)$$

$$\begin{aligned}\text{LOBR2LOCR} \rightarrow\ & \text{lobr2locr}_1 \quad (\ \text{lobr}_1\ ,\ \text{lobr}_2\ ,\ \text{lobr}_3\ ) \quad (0.8)\ | \\ & \text{lobr2locr}_1 \quad (\ \text{lobr}_4\ ,\ \text{lobr}_5\ ,\ \text{lobr}_6\ ) \quad (0.2)\end{aligned}$$

In the above grammar, *Start* and *LOBR2LOCR* are nonterminals, the numbers in parentheses are the usage probabilities of the rules that they follow, and finally, *'lobr2locr$_1$'*, *'lobr$_1$'* ... *'lobr$_6$'*, *'camera'*, *'('*, *')'* and *','* are terminal symbols.

6.2.2 Modeling Service Composition as a PCFG

In this section, we provide the details of how the PCFG can encapsulate service composition information and patterns specific to a sensor network instance with predefined set of services integrated on the sensors. Based on the way to describe service compositions as strings (Section 6.2.1), we describe how a PCFG that describes a language for the efficient compositions can be constructed from a set of strings representing previous compositions. We also decribe how frequent subcompositions (or composition alternatives) can be inferred during the PCFG construction and how to use them to improve the service composition construction.

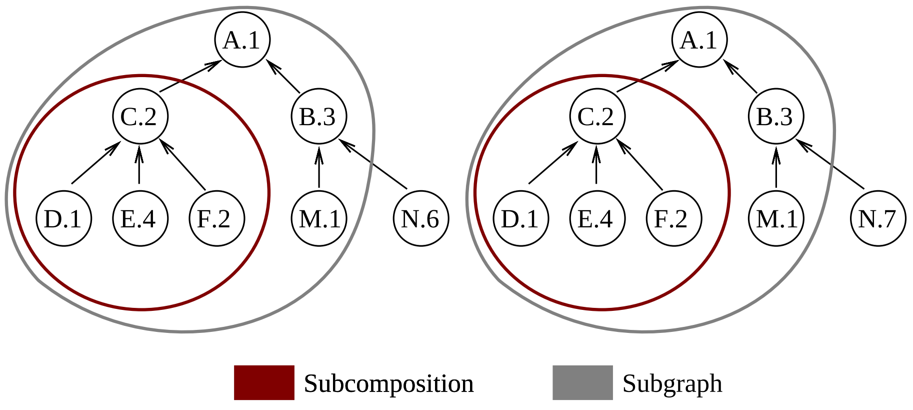
Although there is a lack of previous efforts on learning the service compositions in sensor networks or web applications, a similar problem of learning in component based systems has been widely studied in recognizing the causes for system errors. The most relevant paper on error detection that also utilizes PCFGs, was written by Kiciman et al. [50]. It presents a system called *Pinpoint* that detects faults in the application layer of internet services. The detection algorithm constantly monitors software component interactions. The training stage of the system provides Pinpoint with steady-state behavior of the system while in the run-time, faults are recognized due to behavior that is too divergent from the general case. The first of the two main methods used in Pinpoint is the weighted graph that models the frequency of the component (service) interactions in the system (each component is a vertice in the weighted graph). The second method uses PCFGs to model the order in which other services are used. Hence, every service is a nonterminal that points to an ordered set of other services. In this section, we also give a simple example of advantages resulting from our methodology compared to [50]. This is to show that globally available composition information for the system allows for generation of PCFGs with better composition capabilities.

We finalize the section with the method of regenerating service compositions according to the availability of services during the application lifetime, and what advantages such regeneration provides compared to our previous efforts on automated composition [119] (see Section 6.1).

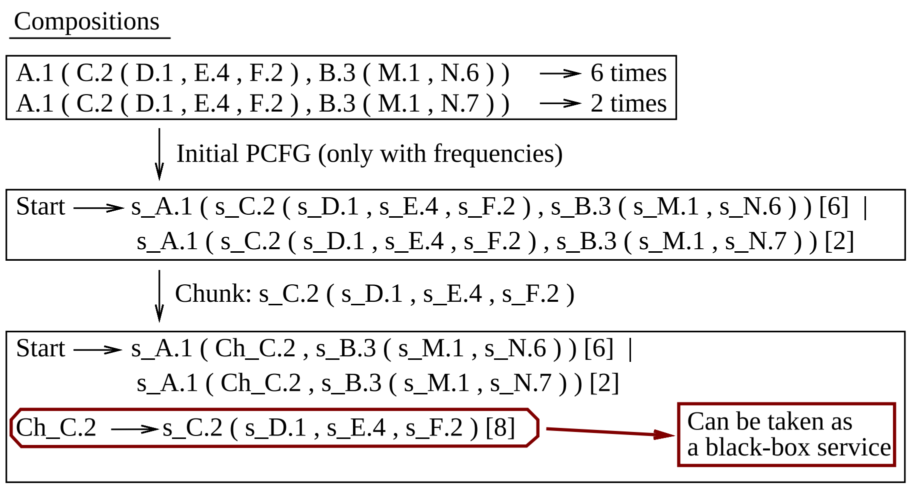
6.2.2.1 Finding Subcompositions and Alternatives via PCFG Inference

In our PCFG inference efforts, we have adopted the usage of a *chunk* operator as a means of shortening the constructed grammar. This operator basically looks for frequently appearing patterns in the grammar and replaces them with a new nonterminal, which in turn generates this pattern. In this section, we are using a similar methodology to find subcompositions which are used frequently, so in practice, it would be advantageous to encapsulate them as black-box services.

A subcomposition is a part of the connections and services that are utilized for a composition. Furthermore, we assume that any subcomposition describes how an intermediate service is composed down to the source services (i.e. services that do not require any input from other services) and is complete (i.e. all inputs and outputs are satisfied). This is the main difference of the frequent subcomposition inference from *graph mining*. We do not find a frequent subgraph of the previously seen composition graphs, but rather we find a frequent composition scheme of one of the services used in previously seen composition graphs (down to services that do not require any input from other services to execute). The example in Figure 6.14a shows two types of frequent structures that have been marked in two composition schemes for the service *A.1*. Elevating a subgraph as a subcomposition depends on



(a) Subcomposition Example

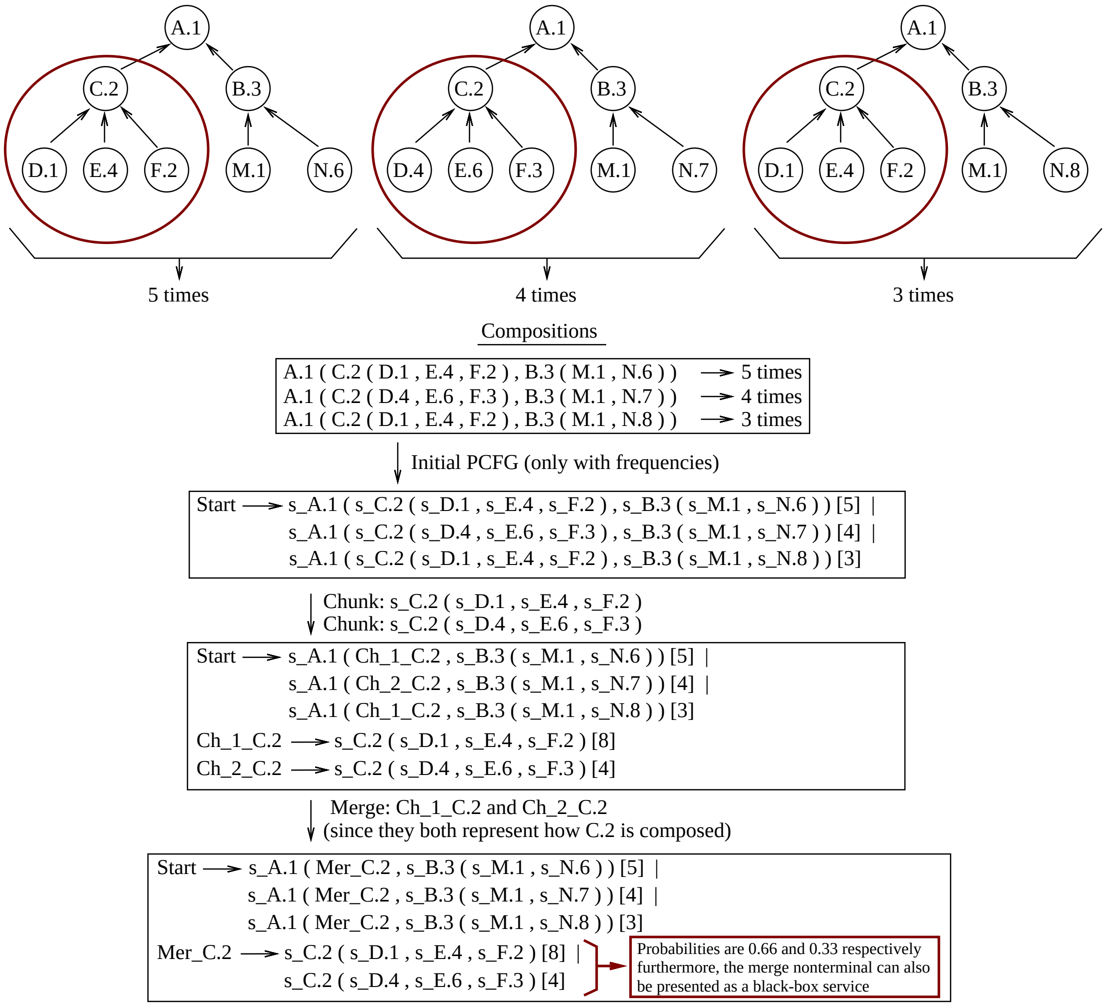

Compositions
A.1 (C.2 (D.1 , E.4 , F.2) , B.3 (M.1 , N.6)) → 6 times
A.1 (C.2 (D.1 , E.4 , F.2) , B.3 (M.1 , N.7)) → 2 times
Initial PCFG (only with frequencies)
Start → s_A.1 (s_C.2 (s_D.1 , s_E.4 , s_F.2) , s_B.3 (s_M.1 , s_N.6)) [6] |
s_A.1 (s_C.2 (s_D.1 , s_E.4 , s_F.2) , s_B.3 (s_M.1 , s_N.7)) [2]
Chunk: s_C.2 (s_D.1 , s_E.4 , s_F.2)
Start → s_A.1 (Ch_C.2 , s_B.3 (s_M.1 , s_N.6)) [6] |
s_A.1 (Ch_C.2 , s_B.3 (s_M.1 , s_N.7)) [2]
Ch_C.2 → s_C.2 (s_D.1 , s_E.4 , s_F.2) [8]
Can be taken as a black-box service

(b) Subcomposition Discovery by the Chunk Operator

Figure 6.14: Finding Subcompositions via PCFG Inference

the frequency of its use.

In the previous subsection, we have described how a composition scheme can be represented as a parenthesized string. In such a language, a subcomposition consists of all the services (hence further subcompositions too) within two matching parentheses. Such a subcomposition describes how the service name preceding the matching parentheses is composed by a set of other services. During the PCFG inference from the set of compositions, we check for the frequency of each subcomposition and compare it to a threshold before assigning to it a nonterminal representing a new black-box service. This threshold is application specific, and depends on the size of the training data. Figure 6.14b shows an example of such a replacement,


A.1
C.2
B.3
D.1
E.4
F.2
M.1
N.6
5 times
A.1
C.2
B.3
D.4
E.6
F.3
M.1
N.7
4 times
A.1
C.2
B.3
D.1
E.4
F.2
M.1
N.8
3 times
Compositions
Initial PCFG (only with frequencies)
Chunk: s_C.2 (s_D.1 , s_E.4 , s_F.2)
Chunk: s_C.2 (s_D.4 , s_E.6 , s_F.3)
Merge: Ch_1_C.2 and Ch_2_C.2
(since they both represent how C.2 is composed)

```
A.1 ( C.2 ( D.1 , E.4 , F.2 ) , B.3 ( M.1 , N.6 ) )   ⟶ 5 times
A.1 ( C.2 ( D.4 , E.6 , F.3 ) , B.3 ( M.1 , N.7 ) )   ⟶ 4 times
A.1 ( C.2 ( D.1 , E.4 , F.2 ) , B.3 ( M.1 , N.8 ) )   ⟶ 3 times
```

```
Start ⟶ s_A.1 ( s_C.2 ( s_D.1 , s_E.4 , s_F.2 ) , s_B.3 ( s_M.1 , s_N.6 ) ) [5] |
         s_A.1 ( s_C.2 ( s_D.4 , s_E.6 , s_F.3 ) , s_B.3 ( s_M.1 , s_N.7 ) ) [4] |
         s_A.1 ( s_C.2 ( s_D.1 , s_E.4 , s_F.2 ) , s_B.3 ( s_M.1 , s_N.8 ) ) [3]
```

```
Start ⟶ s_A.1 ( Ch_1_C.2 , s_B.3 ( s_M.1 , s_N.6 ) ) [5] |
         s_A.1 ( Ch_2_C.2 , s_B.3 ( s_M.1 , s_N.7 ) ) [4] |
         s_A.1 ( Ch_1_C.2 , s_B.3 ( s_M.1 , s_N.8 ) ) [3]
Ch_1_C.2 ⟶ s_C.2 ( s_D.1 , s_E.4 , s_F.2 ) [8]
Ch_2_C.2 ⟶ s_C.2 ( s_D.4 , s_E.6 , s_F.3 ) [4]
```

```
Start ⟶ s_A.1 ( Mer_C.2 , s_B.3 ( s_M.1 , s_N.6 ) ) [5] |
         s_A.1 ( Mer_C.2 , s_B.3 ( s_M.1 , s_N.7 ) ) [4] |
         s_A.1 ( Mer_C.2 , s_B.3 ( s_M.1 , s_N.8 ) ) [3]
Mer_C.2 ⟶ s_C.2 ( s_D.1 , s_E.4 , s_F.2 ) [8] |
           s_C.2 ( s_D.4 , s_E.6 , s_F.3 ) [4]
```

Probabilities are 0.66 and 0.33 respectively furthermore, the merge nonterminal can also be presented as a black-box service

Figure 6.15: Subcomposition Example

where the example from Figure 6.14a is presented as a PCFG. The numbers in the brackets show how many times the composition has been used. The subcomposition shown in Figure 6.14a is assigned the chunk nonterminal ($Ch\_C.2$), which can later be advertised in the system as a black-box service.

Once the subcompositions are found, the probabilities of alternative subcompositions can be found by the *merge* operator in PCFG inference. As aforementioned, *merge* operator basically combines the rules of two nonterminals into a new nonterminal, and replaces the occurrences of both these nonterminals with the newly constructed nonterminal. During the PCFG construction, this operator combines only nonterminals that are the subcompositions of the same service. Consider the example in Figure 6.15. In this case, two frequent subcompositions for the service *C.2* are combined into a new nonterminal, which can be provided later to the end-user as a single black-box service. The probabilities of the rules are of course directly related to how many times each has been used in the training data.

The advantage of the process of finding subcompositions and the alternatives for such subcompositions are two-fold. First, the efficiency of composing services improves. Being able to detect efficient compositions from previous requests for services (hence never directly composing them) saves processing time. Second, the service space is reduced because certain elementary services which are used only in frequent subcompositions can be removed and treated as separate entities.

6.2.2.2 Advantage of the PCFG-based Service Composition Learning in Sensor Networks

An example in Figure 6.16 illustrates why training a PCFG on service compositions yields better results than the weighted interactions of Kiciman et al. [50]. In the figure, rectangles represent services and the circles represent the sensor nodes while the links between the circles represent the logical communication connections. In the example, the service C_1 uses two types of services to receive its required inputs: an instance of A, and an instance of B, each indexed to distinguish between their different instances. In this sample network, if A_1 and B_1, or A_2 and B_2 are chosen together, then their data flows meet at two nodes (Node_5 and Node_6, respectively), causing congestion. Hence, the efficient compositions use either A_1 and B_2, or A_2 and B_1 together to provide the inputs to C_1. PCFG will store and utilize this pattern. However, approach similar to [50] that stores the edge weights will assign equal usage probabilities to A_1, A_2, B_1 and B_2. Consequently, the usage of pair (A_1, B_2) has the same probability (0.25) as of pair (A_2, B_2) even though the latter causes the congestion, while the former does not.

The remedy proposed by Kiciman [50], is to use the list of other components that are called by a component. Although this remedy works for the given example, it fails with minimal extension of the example shown in Figure 6.17 where services A_2 and B_1 are able to utilize the services D_1 or D_2 and E_1 or E_2, respectively.

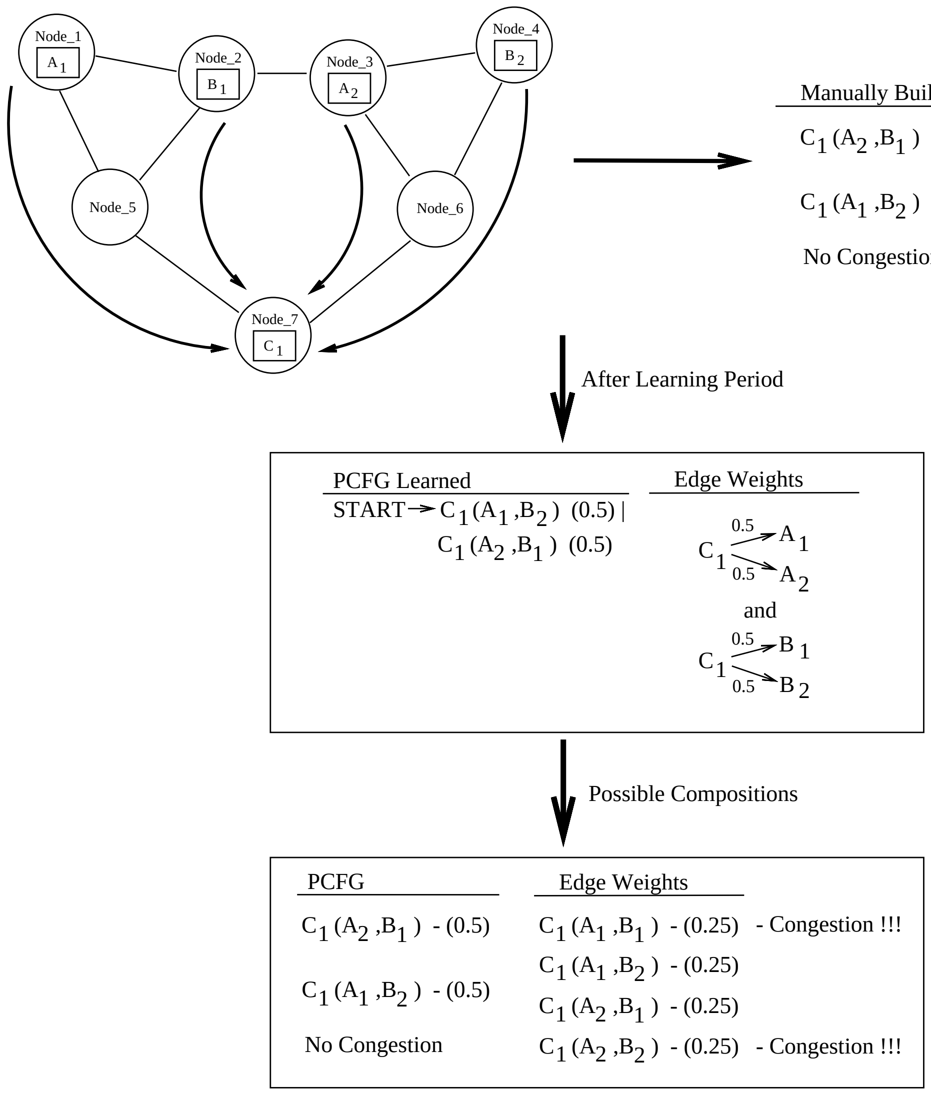
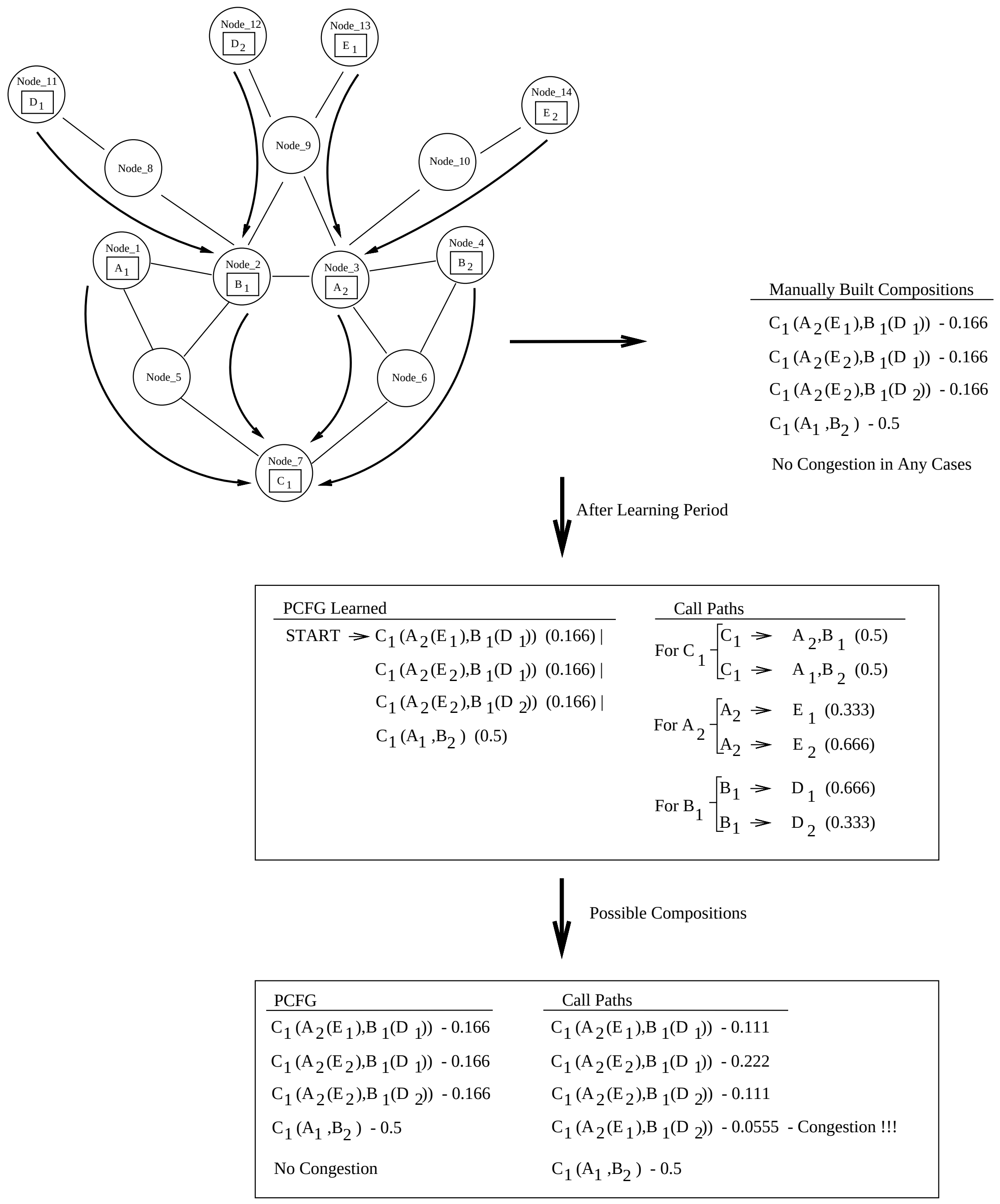


Figure 6.16: An Example to Present the Advantage of the PCFG-based Service Composition over Edge Weights

Since the call paths are only examined at a local level, the remedy will choose a composition with A_2 using E_1 and B_1 using D_2 at the same time (hence causing a congestion at $Node\_9$, since their data flow routes intersect there). However, we train PCFGs from entire compositions, so our method knows that in no efficient composition E_1 and D_2 have ever been used together.

In the previous subsection, we have presented how the *chunk* and *merge* operators help with the discovery of subcompositions and the alternatives for a frequent subcomposition. It can easily be argued that such a process will suffer from the same



Manually Built Compositions

$C_1(A_2(E_1),B_1(D_1))$ - 0.166
$C_1(A_2(E_2),B_1(D_1))$ - 0.166
$C_1(A_2(E_2),B_1(D_2))$ - 0.166
$C_1(A_1,B_2)$ - 0.5
No Congestion in Any Cases

PCFG Learned

START $\to$ $C_1(A_2(E_1),B_1(D_1))$ (0.166) |
$C_1(A_2(E_2),B_1(D_1))$ (0.166) |
$C_1(A_2(E_2),B_1(D_2))$ (0.166) |
$C_1(A_1,B_2)$ (0.5)

Call Paths

For C_1: $C_1 \to A_2,B_1$ (0.5); $C_1 \to A_1,B_2$ (0.5)
For A_2: $A_2 \to E_1$ (0.333); $A_2 \to E_2$ (0.666)
For B_1: $B_1 \to D_1$ (0.666); $B_1 \to D_2$ (0.333)

PCFG	Call Paths
$C_1(A_2(E_1),B_1(D_1))$ - 0.166	$C_1(A_2(E_1),B_1(D_1))$ - 0.111
$C_1(A_2(E_2),B_1(D_1))$ - 0.166	$C_1(A_2(E_2),B_1(D_1))$ - 0.222
$C_1(A_2(E_2),B_1(D_2))$ - 0.166	$C_1(A_2(E_2),B_1(D_2))$ - 0.111
$C_1(A_1,B_2)$ - 0.5	$C_1(A_2(E_1),B_1(D_2))$ - 0.0555 - Congestion !!!
No Congestion	$C_1(A_1,B_2)$ - 0.5

Figure 6.17: An Example to Present the Advantage of the PCFG-based Service Composition over Call Paths

myopic problem of creating suboptimal compositions from which the *edge weights* and *call paths* methodology discussed above suffer. We hereby present that this problem can be avoided by using an entropy-based scheme for applying these operators. Please note that the chunk operator as utilized in this domain finds only the frequent subcompositions, hence has no generalizing effect, and will not create sub-optimal compositions. However, doing the merge may cause such an effect, hence should be used with great care. Let's work with an example to illustrate this issue.

Suppose we have found n subcompositions for the same service S, which have frequencies $f_1 \dots f_n$, and the sum of these frequencies is $\sum_{i=1}^{n} f_i = f_{sum}$. When we combine all these alternative subcompositions into a single new subcomposition nonterminal (by the merge operator), the probability for each subcomposition in this nonterminal (i.e., the corresponding rule probability) becomes $p_i = \frac{f_i}{f_{sum}}$. In this setting, the total probability of a subcomposition being used in place of another subcomposition (i.e. the probability of generalized usage) is:

$$\sum_{i=1}^{n} [(\text{Probability of generating a case where } i \text{ was used}) \quad \times$$

$$(\text{Probability of using subcomposition other than } i)] =$$

$$= \sum_{i=1}^{n} p_i(1-p_i) = \sum_{i=1}^{n} p_i - p_i^2 = 1 - \sum_{i=1}^{n} p_i^2 \quad .$$

This value is the total probability that the PCFG based service composition may use subcomposition $j \neq i$ when in reality i would be used if no generalization had occurred. We want such generalization to have as low probability as possible, hence $\sum_{i=1}^{n} p_i^2$ to be as large as possible. This gives us a bound on which frequent subcompositions are more desirable, and which merges are more beneficial to hold generalization probabilities low while giving the advantage of providing black-box services to the end-user. Basically, we want to find very frequent subcompositions, but we also want subcomposition alternatives where one dominates the others in terms of usage frequency (which increases the value of $\sum_{i=1}^{n} p_i^2$). We evaluate such an approach in Section 6.2.3.

6.2.2.3 Utilization of PCFGs to Generate Compositions

After a PCFG is constructed from previous composition examples, the next step is to utilize it to generate composition graphs (i.e. the graphs which denote how services connect to each other in a composition). For this purpose, we propose

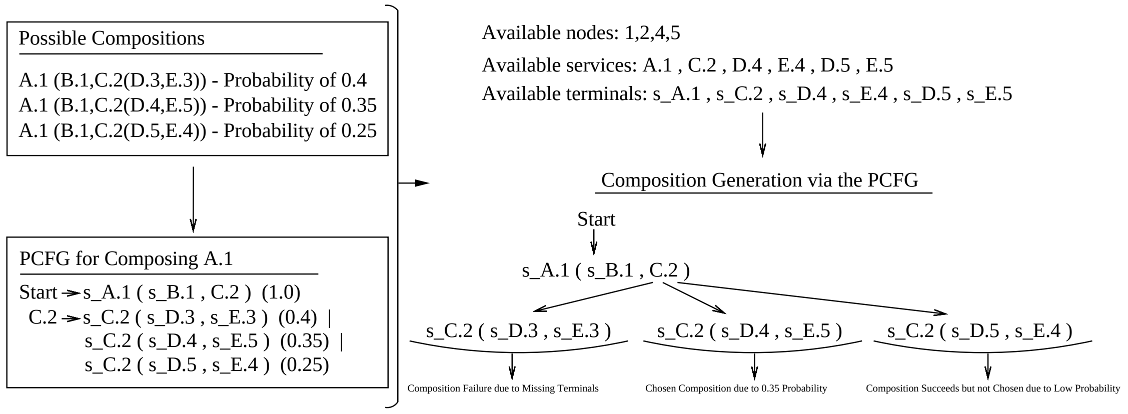

Possible Compositions
A.1 (B.1,C.2(D.3,E.3)) - Probability of 0.4
A.1 (B.1,C.2(D.4,E.5)) - Probability of 0.35
A.1 (B.1,C.2(D.5,E.4)) - Probability of 0.25
PCFG for Composing A.1
Start → s_A.1 (s_B.1 , C.2) (1.0)
C.2 → s_C.2 (s_D.3 , s_E.3) (0.4) |
s_C.2 (s_D.4 , s_E.5) (0.35) |
s_C.2 (s_D.5 , s_E.4) (0.25)
Available nodes: 1,2,4,5
Available services: A.1 , C.2 , D.4 , E.4 , D.5 , E.5
Available terminals: s_A.1 , s_C.2 , s_D.4 , s_E.4 , s_D.5 , s_E.5
Composition Generation via the PCFG
Start
s_A.1 (s_B.1 , C.2)
s_C.2 (s_D.3 , s_E.3)
s_C.2 (s_D.4 , s_E.5)
s_C.2 (s_D.5 , s_E.4)
Composition Failure due to Missing Terminals
Chosen Composition due to 0.35 Probability
Composition Succeeds but not Chosen due to Low Probability

Figure 6.18: An Example on How to Generate a Composition via the PCFGs Given the Set of Available Services

the usage of a centralized scheme where the PCFG is held at a centralized decision maker, where the list of available services are also kept. As it can be seen, the service name together with the id of a sensor node on which it resides constitute a unique terminal symbol for our PCFG. We furthermore assume that the high probability rules represent the fact that certain composition schemes have been preferred over others, hence denote a more efficient composition. We leave the PCFGs where rules are labeled with performance values for future work.

The creation of a composition essentially involves generating a sentence from the PCFG, hence is a modification to the Early-Stolcke parsing [17], [47], where the productions with higher probability are kept, and the productions with unavailable services are directly eliminated. In Figure 6.18, we give an example on how the composition is generated at the centralized decision maker. In this example, there are five nodes and five basic services (A, B, C, D and E). We denote an instance of a service at a specific node with a dot, e.g. D.5 means an instance of service D residing on node 5. Each such instance is made into a terminal in the presented grammar with the '*s_*' in front of it, to distinguish between terminals and nonterminals. As it can be seen, although the composition with probability 0.4 is more desirable (due to previous usage frequency, also represented in the grammar), we cannot create it since node 3 and all of the services on it are unavailable, and the creation of such composition from the grammar with the related terminals cannot complete. Instead, the composition with 0.35 probability is selected. Please note that nodes need to exchange messages to update the list of available nodes, as described in Section 6.1. The benefit of using the PCFGs here are three-fold. First, we already have the probabilities of use of each service in the past that represent the preference for using them currently. Second, the exchange of messages between services with low preferences can be eliminated. Finally, metadata information can be limited to indication if a node is available; we do not need all input/output lists since we already know the links between services. This is a significant reduction compared to an input/output matching based automated composition method, hence improvement of both communication overhead and security of composition construction.

6.2.3 Evaluation

In this section, we demonstrate two advantages of the PCFG based composition. First, we will show that by changing the lower bound on the generalizing threshold (as defined in Section 6.2.2.2), the number of the composition alternatives that are substituted in the created composition changes. More precisely, the lower the generalizing threshold, the more likely it is that the optimal subcomposition scheme will be replaced with another one. From that point of view, the methods of *Edge Weights* and *Call Paths* can be seen as generalizing without any such threshold. We present the effect of the value of generalizing threshold in comparison to *Call Paths* methodology in which generalizing is a bit more suppressed in general.

We also evaluate how the PCFG based composition helps in reducing processing during the composition creation, as compared to an automated composition scheme from Section 6.1. We have already discussed how the identification of subcompositions creates black-box services for the end user. We evaluate how the generalization threshold affects the faster composition (since the lower the generalization threshold, the more black-box services there are and the less time it takes to compose the solution).

We start with the description of the simulation setting with an example application and then we provide the described above evaluations for the PCFG based composition on the output from this simulation.

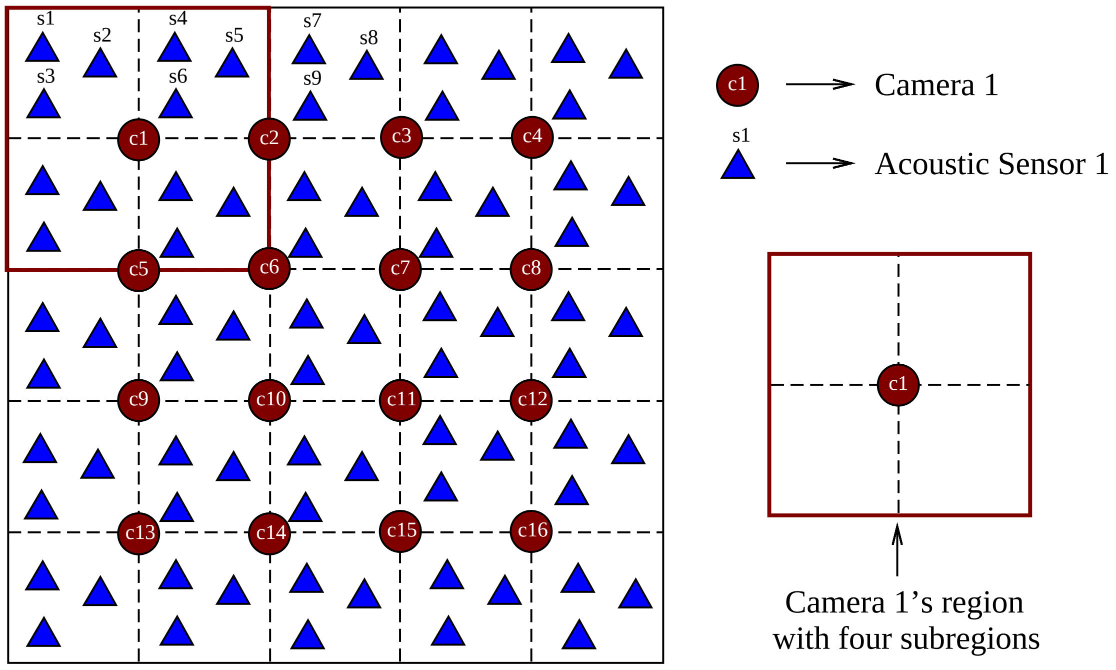


Figure 6.19: Simulation Application

6.2.3.1 Simulation Setting

For our evaluations, we have implemented a simulation as a simple version of the application in Figure 6.13. See Figure 6.19 for a detailed presentation of this scenario. We have a 5x5 grid where each 2x2 subgrid is in the range of a camera sensor, hence covering an area of four grid rectangles. Each camera connects to three acoustic sensors (one from each rectangle, hence no acoustic sensor is chosen from one rectangle) to detect the location of a potential target. There are 16 2x2 subgrids, hence 16 cameras, and each subgrid rectangle (there are 25 of them) has three acoustic sensors (75 in total) for cameras to choose from.

Each user request in our simulation is for 2 to 4 camera service's composition, while the cameras have a preset probability for using each acoustic sensor in each subregion. We have set the most likely acoustic sensor in a subregion with probability between 0.85 and 0.95 (p_1) for each camera (i.e., each camera separately preassigns a likelihood to each acoustic sensor in each subregion). The other two sensors are assigned probabilities from the remaining probability range (with total probability of p_2+p_3=1-p_1). We assume the processing cost for choosing an acoustic

```
START -> u_3_12 ( M_c3 , M_c12 ) (0.005)
M_c3 -> c3 ( s7 , s11 , s27 )  (0.26)  |
        c3 ( s7 , s22 , s27 )  (0.23)  |
        c3 ( s11 , s22 , s27 ) (0.21)  |
        c3 ( s7 , s11 , s22 )  (0.30)
M_c12 -> c12 ( s45 , s56 , s59 ) (0.31) |
         c12 ( s40 , s45 , s59 ) (0.22) |
         c12 ( s40 , s56 , s59 ) (0.22) |
         c12 ( s40 , s45 , s56 ) (0.25)
```

Figure 6.20: A Sample from the Constructed PCFGs

sensor in a subregion to be three units (i.e. we assume a selection process) for an automated composition scheme. For the PCFG however, we take the number of symbols seen at the first level from the *START* nonterminal, i.e. each black-box service is only given a processing cost of one.

6.2.3.2 Results

In our simulations, we used 10 cases of the simulation scenario given in the previous subsection. Each of these cases contains 1000 user requests, which chooses 2-4 camera services to compose. Later, we use the outputs of these simulations, and generate PCFGs with our proposed method. We changed the allowance of generalization for service's subcomposition to be chosen as a black-box, as given in Section 6.2.2.2. We have asserted in our experiments that each service should occur at least 20 times in the training data to be chosen as a parameter to the *chunk* or *merge* operator, regardless of its generalization threshold. This threshold is application specific, and is kept constant in the experiments since it is not worth representing a service as a black-box abstraction if it is barely used in the application.

A sample of the PCFG rules constructed from the output of our simulations is presented in Figure 6.20. In the figure, the *START* rule produces a user request from the cameras 3 (*c3*) and 12 (*c12*), i.e. *u_3_12*. This request uses the black-box representations of camera services 3 and 12, which are represented as nonterminals constructed by the merge operator as previously demonstrated (i.e. *M_c3* and *M_c12* where *M* represents the merge). Although the black-boxes include more rules in the

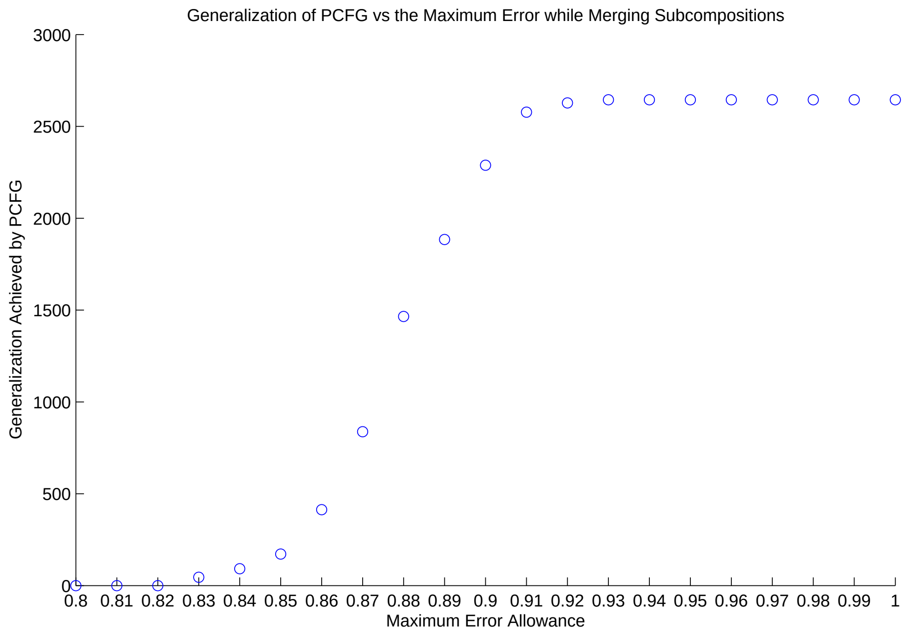


Figure 6.21: Generalization Results

exact output than presented here, we have shown the highest four, and normalized them to reach a sum of probabilities equal to 1.0, again for presentational purposes. In the rules, a camera service uses three acoustic sensors (s followed by the id, as shown in Figure 6.19); this is represented as a string according to the composition representation language (see Section 6.2.1). The number in parentheses following the rule denotes rule's usage probability (due to the composition frequencies) compared to the other rules in the same nonterminal, as calculated from the training data.

Figure 6.21 presents the generalization achieved on the training data by varying the allowed generalization metric (1-*generalization threshold*). This metric can be at most 1.0, meaning there are infinitely many rules each with probability going to 0 (e.g. $1 - \sum_{i=1}^{\infty} p_i^2 = 1$). We present the results with different generalization allowance in range [0.8, 1.0], since we have found out that there are no possible merges until 0.83 (this can be observed from the figure since generalization is 0 until x-axis is 0.83). The y-axis in the figure represents how many subcompositions in training data would be substituted with an alternative one, and increases with the generalization allowance. The method of *Call-Paths* [50] is the point where the allowance is 1.0 (i.e. all generalizations are allowed, hence every service subcomposition is a black-box). The automated composition [119] is the one with the generalization allowance of 0.0, hence results in the generalization of 0 (which is achieved by the PCFG method until 0.83 generalization allowance).

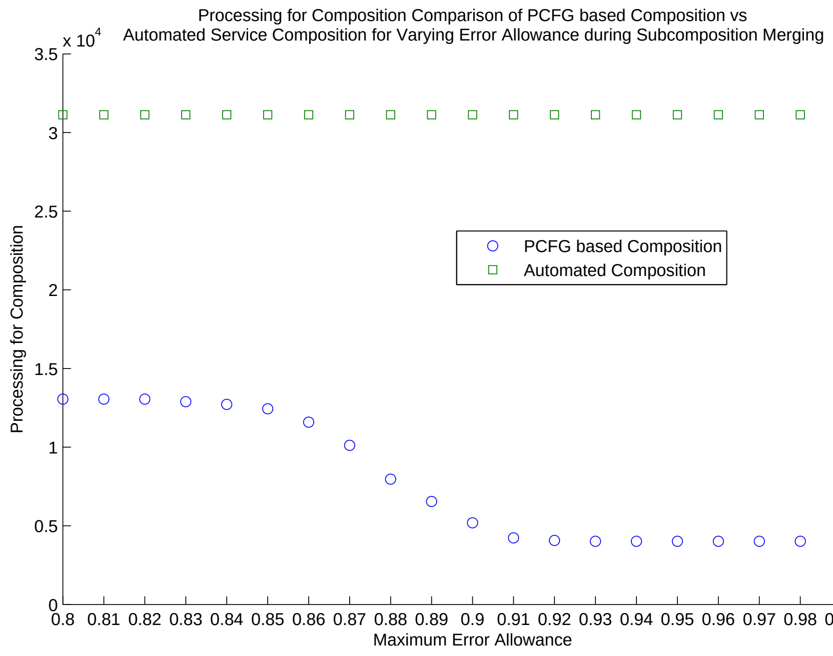


Figure 6.22: Processing Cost Results

In Figure 6.22, we provide the results showing the composition creation processing cost improvements obtained by the PCFG based composition using black-box service descriptions as a function of the generalization threshold. As mentioned before, increased generalization threshold allow more mergess that combine more alternative service subcompositions into a single nonterminal (black-box representation). We have assumed that the automated subcomposition chooses one of the three acoustic sensors for each subregion when a camera service is activated, hence brings a three-unit processing cost (for each acoustic service). In contrast, when the PCFG creates a black-box representation for this camera service, the corresponding cost incurred is just one-unit. The figure clearly shows that the increased generalized threshold helps with lowering processing cost of composition. Without the generalization (i.e., when generalization threshold is less or equal to 0.83 and all merges are suppressed), the PCFG method works only through its rules, and hence brings down the cost of choosing each camera or acoustic service to one (for acoustic services, the automated composition incurs the cost of three-units since we assume an evaluation of selection the best acoustic sensor among the three possible ones in the given subregion). Furthermore, the method of *Call-Paths* [50] is represented by the point where the x-axis has 1.0 allowance, hence every camera service is made into a black-box representation of its subcomposition.

We have shown through the simulations that the PCFG based method achieves lower processing cost for creating compositions than *Call-Paths*. We furthermore demonstrated that the fine tuning of the application specific generalization threshold can resolve a trade-off between processing cost and composition generalization, as expected.

6.2.4 Related Work in Service Composition Learning

As already mentioned, we are not aware of any previous efforts on learning the service compositions in sensor networks or web applications as an automated composition generation method by learning from previous compositions. A similar problem of learning in component based systems, however, has been widely studied in systems recognizing the causes for system errors (see surveys in [161]-[163]). We have already discussed an important work [50], in Sections 6.2.2 and 6.2.3, which we also use for evaluating our approach. We now review similar studies within the same domain.

The authors of [164] introduce *Gingko*, a mechanism which allows users to correlate *casual paths* (the path between components that a message follows) with the errors that occur within a system. Such correlation can be used to detect the root cause of an error in the system. Another work [165] makes use of *Decision Trees* in order to detect failures in a shopping site. The authors train a decision tree on what types of user requests (i.e., interaction with services) cause a failure. C4.5 [166] algorithm is used for this purpose.

Principal Component Analysis (PCA) has been utilized in [167] to classify the frequency of component interactions in internet services into normal and anomalous behavior. Another significant work [168] makes use of variable length *n-grams* to model call and return order between components in distributed systems. In our opinion, such a method is not suitable for the service model since in services composition, concurrent services are called and return data at the same time (e.g. multiple services providing different outputs to the same service at the same time).

Other related error detection techniques use user logs of the visited internet pages [169], CPU-instructions and function-calls [170], and messaging between components [171]. In sensor networks domain, [172] examines simple metrics on network performance.

6.2.5 Conclusions and Future Work

In this section, we have presented our work on the utilization of PCFG modeling for service composition in sensor networks. We demonstrated how different compositions can be represented as strings, which are then used to construct a PCFG. This PCFG can later be utilized to generate compositions at the centralized decision maker, taking into account which services are currently active in the system. We have discussed advantages of our scheme over previous such efforts, and presented simulation results which evaluate how the cost and the generalization performance of service composition is affected by parameter setting in our methodology.

We conclude that the PCFG modeling is an efficient way of gathering and storing composition information; it provides manageable generalization in the constructed compositions, and generates a lower processing overhead compared to systems in which such composition structure is not held. Furthermore, we believe that other domains, such as error detection in software, can also benefit from this approach. We leave this direction as future work, along with a PCFG modeling where the rules are labeled with performance metrics. Moreover, we plan to do a more detailed evaluation of this methodology, especially in network-based metrics such as communication overhead.

6.3 Switch Options for Pervasive and Mobile Applications

This section aims to optimize selection of services in sensor networks and operational mode switching in pervasive applications, and we present the usability of switch options for this purpose. *Switch options* (for more details, see Chapter 10 in [177]) are a special form of real options that can be used to model the value of keeping multiple alternatives available. This applies to valuing general *process flexibility*, defined as the ability to switch among alternative inputs, as well as *product flexibility*, which refers to the ability of manufacturing multiple products in response to changing market demands. When switching costs are absent, exercising the option affects only the current payoff but not any subsequent (switching) decisions. We use switch options to dynamically adjust the efficient interconnection of basic services to changing network conditions. We furthermore demonstrate the switch options approach in a domestic pervasive security application. We describe how false security-breach alarms can be prevented by cleverly switching between secure/insecure modes with an expectation of gains/losses for this application.

An exposition of the theory of switching in virtual organization may be found in [178]. Switch options have been applied to the management of different modes of operation (e.g. ability to produce different materials according to market conditions) for investments [179], [180]. The first of these two papers utilizes a stochastic dynamic programming to encapsulate the value of flexibility to switch between the modes. The second paper examines the option to change the quantity of resources used for production in response to changes in the product demand.

The rest of this section goes as follows. We first describe our methodology for the switch options, and explain how the value of various alternatives can be found. We then evaluate the proposed methodology in two applications: (*i*) dynamic service selection in sensor networks, and (*ii*) switching operational modes in a pervasive security application.

e.g. switching among various types of fuel that a factory may use for its operation

6.3.1 Switch Options Methodology

Switch options constitute a promising modeling and decision making approach for service and operation mode selections that are made during the pervasive application's operation, in the presence of volatile network and environmental conditions. However, the application of this method requires a model for quantifying the value of an alternative service or operating mode.

6.3.1.1 Quantifying the Value of a Selection:

The value of a selection can be measured by the difference between its benefits and the costs incurred by its execution. In the case of service selection in sensor networks, the benefits rely on *Value of Information* (VoI) that is application-specific and depends on the importance, quality and security of a service's output [181]. On the other hand, costs of accessing a service include any energy spent in processing and communication with the provider of the service, as well as the delay for the transmission of the output that it provides. Hence, the *value* (V) of a service S is:

$$V(S) = V_{\text{inf}}(S) - \alpha(t)E_s(S) - \beta(t)D(S) \ , \tag{6.1}$$

where V_{inf} represents the VoI of the output that S produces, E_s represents the energy that is spent by this service, and D is the time it takes for the output of S to reach the requesting service. $\alpha(t)$ and $\beta(t)$ act as unifying parameters for the different units of the above components. They are also application specific and describe the relative importance of energy and delay to the application at a specific point t in time. For example, if the application becomes time-critical, the β value will increase to penalize the service instances with high delay. Similarly, for services with low energy left in the sensor on which they are implemented, the α value will increase. Furthermore it is entirely possible that the information value ($V_{\text{inf}}(S)$), energy spent ($E_s(S)$) and the delay ($D(S)$) vary during the operation; hence they are also time dependent, although this is not explicitly shown in the equation. In this work we are dealing with a simplified, linear valuation model but other valuation techniques can also be applied, a task that we leave for future work. Furthermore, in the case of switch options in other domains, different types of costs are more relevant. We will discuss these in more detail for the application of switching options in the pervasive security applications.

6.3.1.2 Switch Options Modeling

The value of being able to switch to different operating modes or actions during the development of a project is the extra value that can be gained once the switch is exercised. For example, for two operating modes A and B, of which the former is more valuable at the beginning of the project, the extra value of keeping B as an alternative, as long as there are no switching costs, is $V_{\text{Option}} = \sum_{t=t_0}^{t_n} V_{A\to B}(t)$, where $V_{A\to B}(t)$ is the expected value of the extra gain from using B instead of A when market conditions suggest so, and time series $t_0 \ldots t_n$ denotes the times at which the switch was made. Of course, this value is obtained only if choosing B is expected to be more beneficial than keeping A, otherwise the switch will not occur.

The above approach can readily be applied to sensor service selection or pervasive applications. However, it becomes apparent that the network and environmental conditions that drive the switching decisions should be assessed before any selection of sensor service instances or operating modes can take place. We name this task the *test phase*, which is followed by the actual *selection* and *execution* phases. These phases are discussed in the remainder of this section.

Test Phase: once the possible sensor services or the operation modes that provide the necessary functionality are detected, the test phase executes all alternative selections either all at once or one after the other, to estimate the costs that they incur and their value of information. The test run lasts for a set period of time, the length of which determines the cost of performing this evaluation. Often, the conditions for switching between the instances may require that the possible selection alternatives are run all at once.

The cost of a selection is determined during the test phase by: (i) accumulating the energy consumption and delay for processing and communication of the information packets that are relayed along the path that connects the service and the user, for the domain of service selection, and (ii) accumulating monetary and manual labor costs, for other application domains (such as modes of operation in a security application). The value of information that is provided by a selection is also assessed during the test phase, thus the value $V(S)$ (as in Eq. 6.1) can be computed.

From the above discussion, it is apparent that the test phase (and our proposed method based on switch options described herein) cannot be applied in cases when the sensor service, or a mode of operation, is time-critical and short-lived. For example, there is no opportunity to conduct a test phase when one needs to monitor the break-out of a fire in a forest. On the other hand, for long-lived sensor tasks such as temperature or soil contamination monitoring, the costs incurred by the test period are reclaimed by the future gains of switching from one selection to another.

The gains from different selections are estimated using the measurements of the test phase as follows. Let $C(t)$ be a random variable representing network and environmental conditions at time t that define VoI of options A and B at time t, denoted by $V_A(C(t))$ and $V_B(C(t))$, respectively. Also let I be the random variable representing switching signal to switch from option A to B ($I(t) = 1$) and from B to A ($I(t) = -1$), or no action ($I(t) = 0$), at time t. Most often, the switching signal is a function of $C(t)$, and its quality might depend on which options are active, as only currently active sensors can provide input to the computation of function I. Furthermore, let T denote the duration of the test phase. During that time, we make a series of n measurements of $C(t_i)$ and compute $I(t_i)$, where $t_i = i \times T/n$, as well as $V_A(C(t_i))$ and $V_B(C(t_i))$. Based on these measurements and computations, we create the piecewise linear approximations $v_A(t)$ and $v_B(t)$ of $V_A(C(t_i))$ and $V_B(C(t_i))$ for t in $(0, T)$. Likewise, we create a piecewise constant function $s(t)$ (stating currently active option, either A ($s(t) = 0$) or B ($s(t) = 1$)) defined for $t_i < t \leq t_{i+1}$ as 0 if $i = 0$ (we always start with option A active), or, for $i > 0$, $s(t_i) = \min(\max(I(t_i) + s(t_{i-1}), 0), 1)$, where min and max are used merely to keep the result 0 or 1. Then, the benefit per time unit of having an option to switch from A to B is given by the following integral:

$$V(A \to B) = \int_{t=0}^{T} \frac{s(t)(v_B(t) - v_A(t))}{T} dt \ . \tag{6.2}$$

If amortization cost per time unit of having option B (which includes cost of switching between options as well as cost of measurements necessary to generate switching signals) is lower than $V(A \to B)$, option B is worth having. If c_{AB} and c_{BA} denote the switching costs from A to B and B to A, respectively, then the switching cost between options A to B normalized over time T is given by the following sum:

$$c(A \to B) = \sum_{i=2}^{n} \frac{I(t_i)}{2T} [(1 + I(t_i))c_{AB} - (1 - I(t_i))c_{BA}] \ .$$

Indeed, when $I(t_i) = 1$, we switch from option A to option B, and when $I(t_i) = -1$, we switch back, so at those times we need to add the cost of switching, and $I(t_i)(1+I(t_i))/2$ yields 1 if and only if when $I(t_i) = 1$, and similarly $-I(t_i)(1-I(t_i))/2$ yields 1 if and only if when $I(t_i) = -1$.

Finally, if the cost of measurements of switching signals at time t is $m(t)$, then the cost of measurements per time unit is:

$$m(A \to B) = \sum_{i=1}^{n} \frac{m(t_i)}{T} \ .$$

Often, $m(t)$ is independent of t, so $m(t) = m_c$ and then $m(A \to B) = \frac{n \times m_c}{T}$.

The above mathematical model captures a direct relationship between the signals used for switching decisions and the value of the switch option. The test phase is required for calibrating the values of switching option and of switching signals. Better these signals are, larger percentage of the benefit of switching to the best solution are. Furthermore, the cost of amortization and achievable benefits of switching determine whether the alternative services or modes of operation are worth keeping.

Execution Phase: once the feasible choices for a set of selections are narrowed down based on the lowest estimated cost from the test phase, then the chosen subset of possible selections are executed and the gains are computed according to Eq. 6.1, during the execution phase. Once conditions for replacement of services (or operating modes) are specified, they can be monitored continuously as the operation continues. Moreover, monitoring of the environmental phenomena may offer information about the kinds of events requiring switching. For example, this could be evident in the case of high humidity being detected during the test phase that affects one service more than another. Knowledge of such a result can be used to switch automatically when humidity increases in the monitored area. Note that such effects depend on the environment in which the sensor network is deployed, making the test phase essential. Furthermore, to be able to switch between services or operation modes, it is not necessary to run all alternatives during the execution phase. The environmental conditions that are necessary to decide on switching are monitored separately (which constitutes a cost as given in the above mathematical model). The switching points however are *learned* in the *test phase*, where multiple options are run together to see which is more advantageous at which condition.

6.3.2 Switch Options for Service Selection

A Service-oriented Architecture (SOA) approach [182] to wireless sensor networks abstracts them into a set of software services, each of which provides a well-defined functionality and might be deployed on one or more sensor nodes. Service selection is based on the assessment of the processing and communication costs that are incurred when a service instantiation in a given sensor node is chosen as part of a composition graph. There were previous methods proposed for making an efficient choice of operator for general distributed computations (e.g. [183]-[186]). However, these methods have not taken into account *the operational uncertainty* arising in sensor network deployments, which directly affects any estimates of service costs. Operational uncertainty in sensor networks arises from several causes, including (a) a lack of accurate knowledge of the computational and communication resources of sensor nodes and their residual energy that gets depleted over time, (b) changes in the environmental conditions in which nodes are operating, such as background noise that affects the quality of sensor readings (e.g. audio signals), and (c) the changes in the Value of Information (VoI) that the output of a composite service carries for a particular user at a given point in time. We propose the use of switch options to deal with such uncertainty in sensor services.

To demonstrate the application of switching options theory to the domain of sensor service selection, we first provide a real world scenario, and then present the

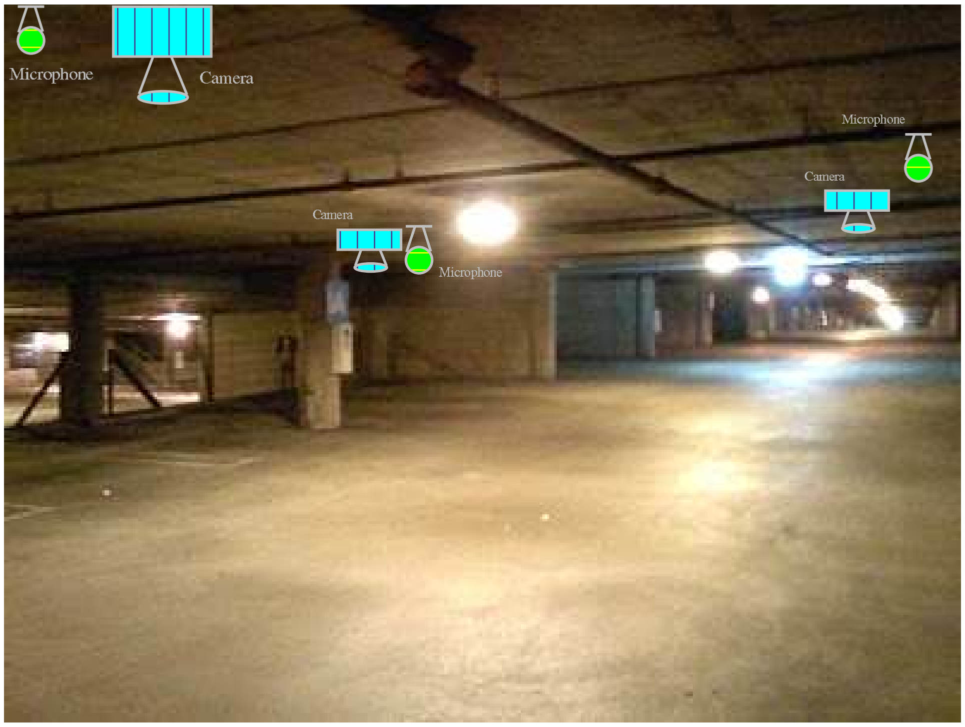


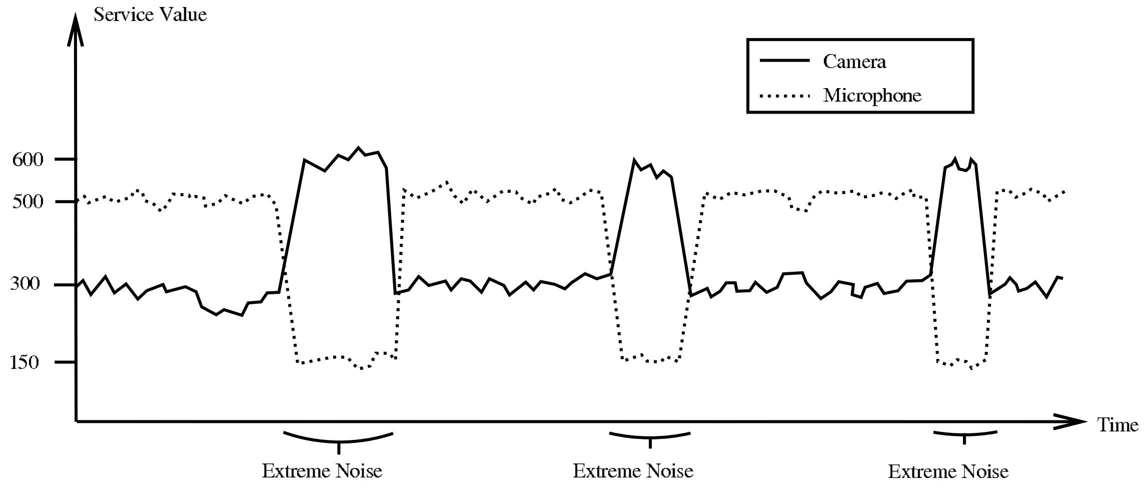
Figure 6.23: A Parking Garage Example to Explain Switch Options Methodology

benefits of the switch options methodology on a simulation based on this scenario.

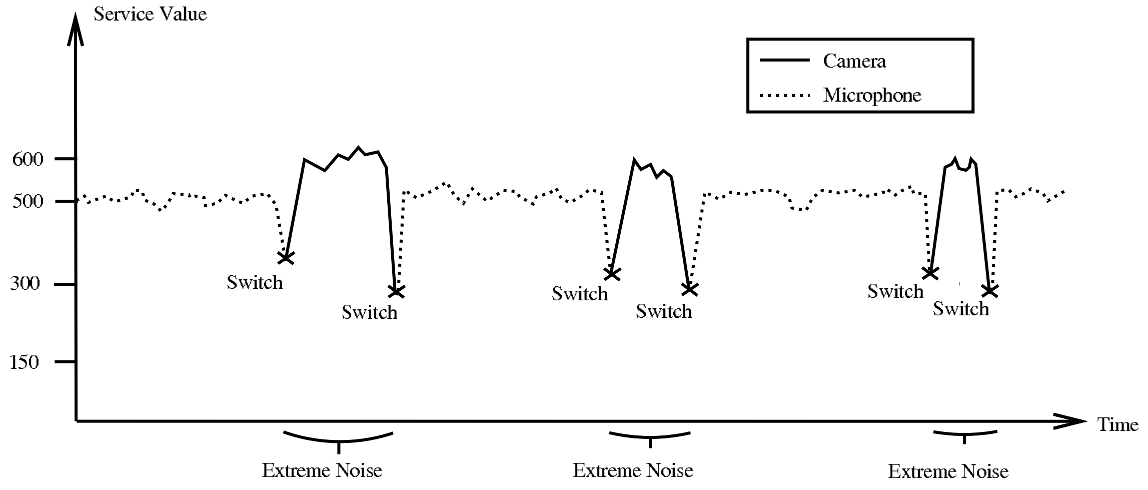
6.3.2.1 Real-World Scenario

As an example real world application of switch options to service selection, we chose a covered parking garage monitoring network given in Figure 6.23. There are two types of services in this network: (*i*) a microphone service of readings from an acoustic sensor to monitor the sound volume, and (*ii*) a camera service that provides views of the area covered by the microphone monitors. We consider a pervasive monitoring application during which automated service selection may choose to utilize one or both of the services in monitoring.

An illustration of expected test period results for this application is shown in Figure 6.24a. The test period measures how external factors affect the VoI and benefits produced by each service. Clearly, the camera view for an area will produce the best results, but running a camera is a costly operation (due to its energy consumption and maintenance costs) in mid-term applications, so the benefit of



(a) Service Value Fluctuations for Two Types of Monitoring Service



(b) Value of Switching Option During the Test Period

Figure 6.24: Expected Test Period Results for Parking Garage Monitoring

using this service is low when no events are happening in the garage. Figure 6.24a also shows that when the VoI of the microphone service drops below a certain level, there is loud noise in the garage, and the VoI provided by the camera service increases. Of course, loud noise often signifies an important event in which case the VoI (the first factor in the above equation) increases even more, giving the camera service a higher benefit than normal. On the other hand, the microphone gives faulty measurements when the noise level is high. This example shows how the switch option balances the *Value of Information (VoI)* provided by the services with the cost incurred by it.

Once the costs of running multiple services in each area during the test period have been incurred, the service selection mechanism can switch between the microphone and the camera service, and can do it with increased efficiency based on information about the conditions that are beneficial for switching gathered during test period. In the long run, the costs of the service test phase is compensated by clever switching actions, which it enables with the information it acquires. Figure 6.24b shows the advantage that could have been gained had switching information been available during the test period shown in Figure 6.24a. The area difference between the curves of Figure 6.24b and the curve of the microphone service (since it is best on average) in Figure 6.24a quantifies the extra value of the switching option.

6.3.2.2 Evaluations

We conduct a simulation-based evaluation of our service selection method via switch options, which is based on the parking garage example described in Section 6.3.2.1. Our goal is to assess the relative gains in service value obtained by using this new approach compared to the naive method of selecting a service based on its current value, as well as the optimal approach that always has complete knowledge of the value of services in a noisy environment. The setup that we simulate includes a microphone and a camera monitoring service for one area of the parking lot; events triggering system responses are set up to investigate how the service value of the microphone or camera changes. A test period was established to measure the switch points according to the sound levels in the environment. These levels are hence associated with the signaling function $I(t)$, as given in Section 6.3.1.2. Here, $C(t)$ (i.e. the environmental conditions) is the sound level in the system at time t. This can be measured by both the microphone and the camera service, since we assume the camera service also encapsulates a microphone; hence the conditions of the environment can be measured by the chosen service during execution phase. This allows for the chosen service signal that it should be switched with the other alternative during execution. In another application with different environmental indicators, extra effort may be needed to monitor conditions which signal the switching of service selections. The cost of this extra effort was also mentioned in Section 6.3.1.2,

Table 6.3: Sound Change Properties for Service Selection Experiment

Type	**Inter-arrival (secs)**	**Sound Level**	**Length of Period (secs)**
Event	exp(mean=100) → av: 100	200+[40 x (Γ(k=90,θ=1/3))] → min: 200, av: 1400	20 x $\mathcal{N}$(μ=1,σ^2=0.1) → av: 20
Non-Event	exp(mean=40) → av: 40	200+[40 x (Γ(k=10,θ=0.5))] → min: 200, av: 400	20 x $\mathcal{N}$(μ=1,σ^2=0.1)] → av: 20

which may make keeping the option infeasible if too costly. While we try to make reasonable assumptions regarding the evolution of noise level in our target environment as well as the characteristics of the process that generates events of interest, our goal is not to exhaustively study all possible statistical distributions and their parameters for these components, but rather gain a qualitative understanding of the performance of these strategies. A realistic assessment of the service selection approaches can only be performed in a deployed sensor network, a task that we leave as future work.

Two experiments have been simulated that differ in the magnitude of the noise level that is assumed in the environment. In our first experiment, the results of which are shown in Figures 6.25 and 6.26, the simulation time is fixed at 100,000 seconds, and the ratio of the test period to the remainder of the simulation time is varied among the runs (we ran 10 cases of 100,000 minutes for each test period length), within the range of values that is depicted on the x axis in the figures. We introduce two types of sound level changes to the environment, on top of the ground noise sound level, which is constant with a value of 200. The first type represents an increase in the environmental noise without any actual event of interest taking place (for example no car driving in or out of the parking lot but outside noise due to nearby construction is increasing). The second type of sound level change that we simulate represents a significant event of interest that occurred in the environment, for example detection of car movement in the parking garage. The properties of these sound level change types are given in Table 6.3.

As discussed above, the value of information (VoI) consists of two components: one that is subjective and describes the utility as assessed by the user, and another one that denotes the objective quality of information (QoI) that data carries. VoI is then represented as the product of these two components. For our experiments,

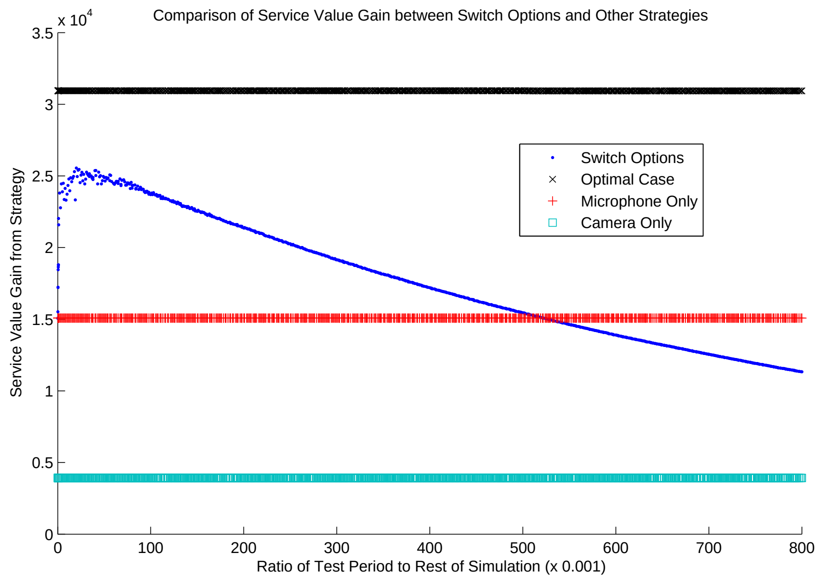


Figure 6.25: Comparison of Switch Options in Experiment 1 for All Test Period Lengths

a constant QoI of 0.9 is assumed for the camera sensor, while the microphone has a QoI that changes according to $1 - (\frac{SoundLevel}{1000.0})^2$, which accounts for loss of quality with high sounds. The utility for the camera sensor service is set to 1.0 upon occurrence of an event, while for the microphone it is constant at 0.2 when no event of interest occurs. Regarding cost, we set it to 0.45 and 0.15 for the camera and the acoustic sensor respectively, which includes both energy and delay as per Equation 6.1. The value $V(S)$ of each service is the computed by the simulator as $V(S) = utility \times QoI - cost$ for each time unit.

In Figure 6.25, the results for service values corresponding to different lengths of the test phase are given for four strategies: the switch options approach, two strategies that select either the camera or the microphone service exclusively, and the optimal approach that has complete and accurate knowledge of the VoI and the costs. Evidently, choosing either the microphone or the camera service provides less value than switching between them during the system's run time. This is to be expected, since a system that does not make use of the test phase does not know if and when it should switch between alternatives. Such decisions in our experiments are made via the expected value by each service at a given sound level. Although

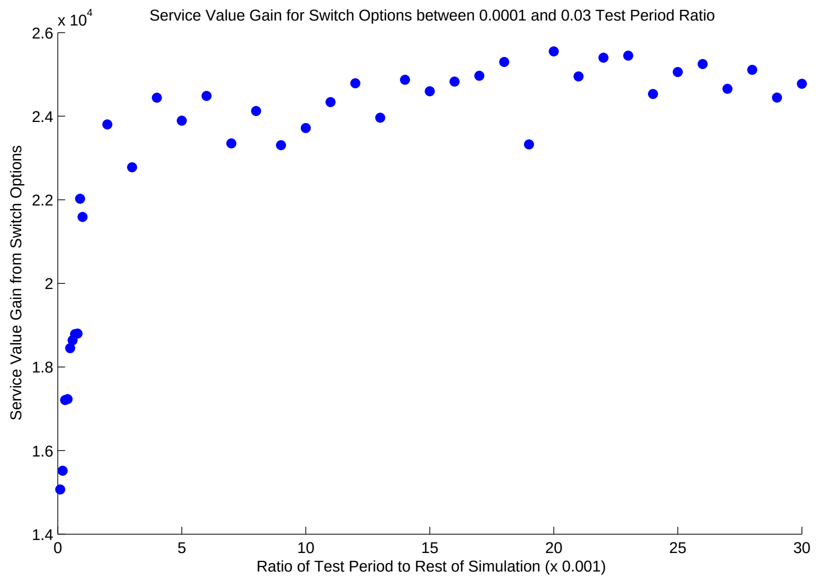


Figure 6.26: Gain from Switch Options in Experiment 1 for Test Period Lengths up to 3% of the Rest of Simulation Period

the test phase is indeed useful, as it gets longer, the value gains decrease due to the excess cost of running (and testing) both services. Figure 6.26 shows the results of a closer look to the varying lengths of the test phase to see the value that peaks the gain for this setting (via comparing them with shorter ones for the same experiment). From the graph, it appears that when the length is too short (for example only 0.1% of the actual system run time), the gain in value is rather small. This is due to the switching options approach not being able to determine when to make a switch, as a consequence of limited experience with events.

The second simulation experiment that we ran, named *Noisy Parking Garage*, differs from the previous one in that the magnitude of the first type of sounds level changes (i.e. those that correspond to changes in the ambient noise level of the environment but not to events of interest) exhibit a much higher average. The interarrival times and the other parameters listed previously are kept the same, but now each such sound level change follows a gamma distribution with $k = 40$ and $\theta = 0.5$ (thus mean is 20 and variance 10), which is again multiplied by 40 and added to ground noise sound level of 200 as before. In this configuration, the ambient sound changes are of high magnitude, and due to the decrease in QoI of

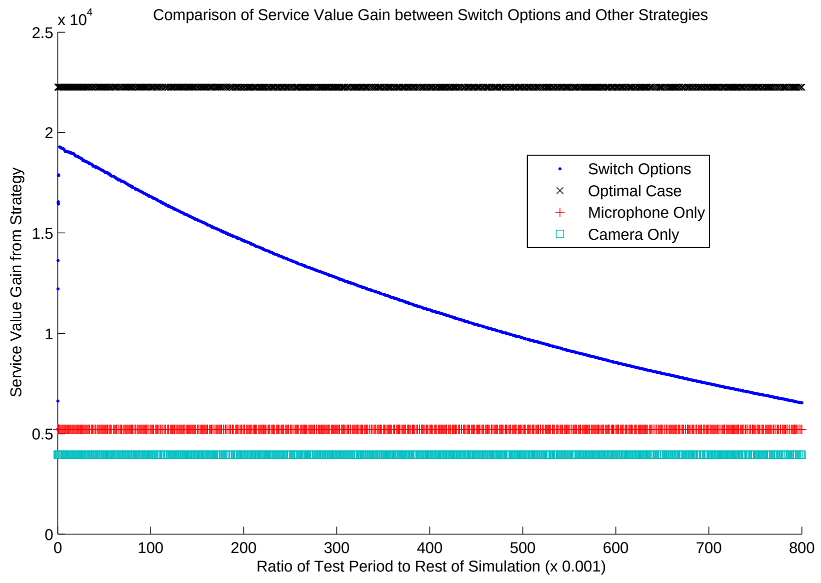


Figure 6.27: Comparison of Switch Options in Experiment 2 for All Test Period Lengths

the microphone service, the total gain from using it is expected to be lower. The results of this simulation setup are presented in Figures 6.27 and 6.28.

As it can be seen from Figure 6.27, the value from the microphone service decreases significantly compared to the previous experiment, while that of the camera service remains the same. This is due to the the higher ambient noise levels of the parking lot that adversely affect the quality of acoustic measurements obtained from the microphone, which decreases its QoI. The overall value obtained through the switch options strategy is similarly decreased, with its peak level becoming lower and the rate of decay higher as the length of the test period increases. Similar results are observed for the value of the optimal service selection.

Finally, in Figure 6.28, we examine the service value obtained through the switch options strategy by ranging the test period length from 0.1% to 3% of the rest of the simulation time. The peak value of the switch option strategy is reached when the test period is set to 0.3% of the overall simulation time, at which point the best tradeoff between discerning switching points and keeping costs minimal is achieved for the given simulation settings.

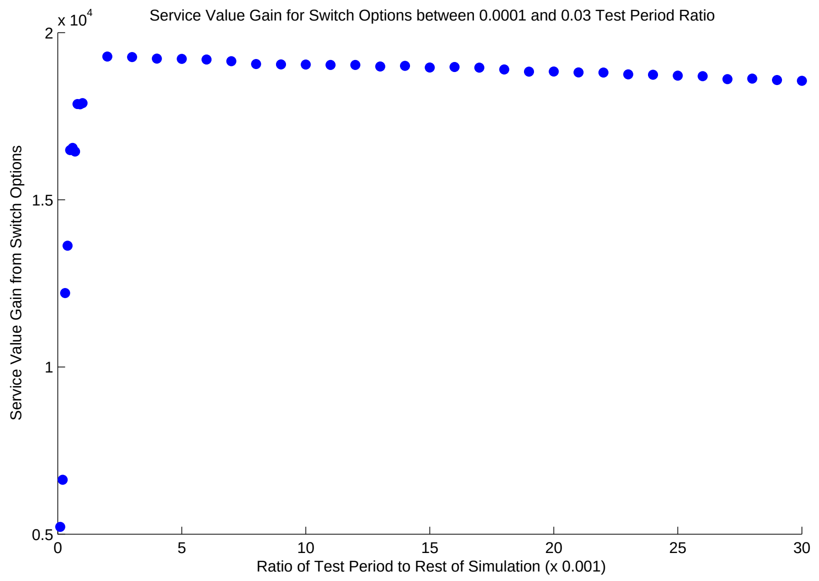


Figure 6.28: Gain from Switch Options in Experiment 2 for Test Period Lengths up to 3% of the Rest of Simulation Period

6.3.3 Switch Options for Pervasive Security Applications

In this section, we consider another application of switch options, the selection of operation modes in a pervasive security system. A house, a gallery or any other secure space where certain valuables are stored are often monitored by a security firm which charges a certain amount of money per time for its services when the security system is in alarm mode. We assume that in addition to the per time fee, the security firm charges also switch costs, a fixed cost fee charged when the alarm is raised and the response crew is dispatched to the monitored area. This is a perfect case for applying switch options because being able to set good indicator values of when to alarm the security firm is crucial to protecting valuables with acceptable cost of protection. False alarms cost money for security charges, but missed security breaches result in the loss of valuables. This increases the importance of *test period* during which the alarm indicators are learned.

We assume a security application where a microphone is used to measure sound levels in the system. The changes of sound levels in the monitored area indicate a potential security problem (e.g. theft). We simulated the security application for 100,000 minutes, and again introduce two types of sound level changes to the system:

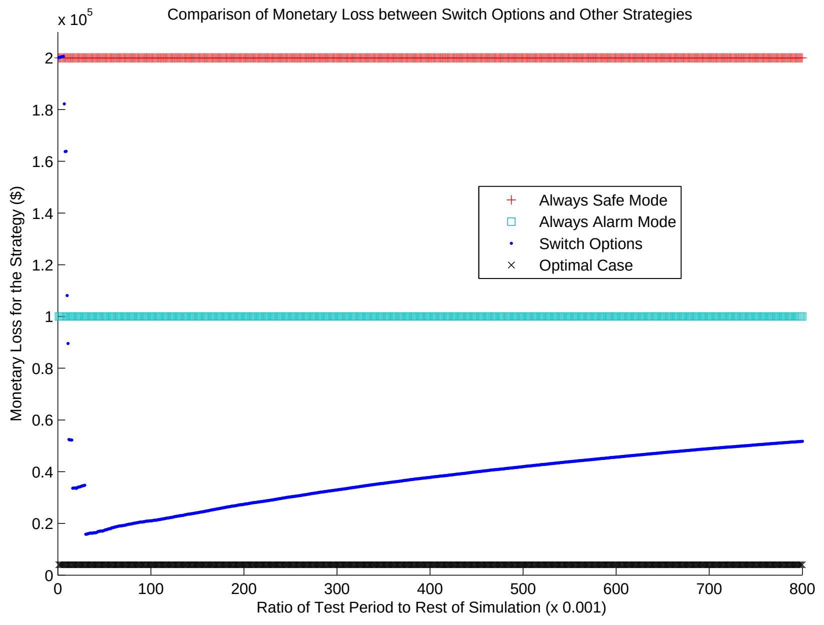


Figure 6.29: Monetary Loss for All Test Period Lengths

(i) security breach, and (ii) safe situation whose properties are detailed in Table 6.4.

Table 6.4: Security Application Sound Change Parameters

Type	Inter-arrival (min)	Sound Level	Length of Period (min)
Security Breach	600+exp(mean=600) $\rightarrow$ min: 600, av: 1200	200+[40 x (Γ(k=90,θ=1/3))] $\rightarrow$ min: 200, av: 1400	5+[35 x $\mathcal{N}$(μ=1,σ^2=0.1)] $\rightarrow$ min: 5, av: 40
Safe Event	300+exp(mean=300) $\rightarrow$ min: 300, av: 600	200+[40 x (Γ(k=20,θ=0.5))] $\rightarrow$ min: 200, av: 600	5+[15 x $\mathcal{N}$(μ=1,σ^2=0.1)] $\rightarrow$ min: 5, av: 20

The costs of running the alarm operating mode is set in our simulation as follows. According to the sound level in the area as an indicator, the security firm charges $50 for switching to alarm mode (i.e. initial service fee), and $1 for each minute that it checks upon the monitored area. Furthermore, we have set the total value of valuables in the monitored area to be $200,000; hence, the losses amount to this value if the switch to alarm mode does not happen during a security breach. These values stay the same during the test phase, meaning that for each minute spent in test phase, the security firm charges $1 to learn the parameters for the indicators (i.e. the sound levels that indicate a security breach, hence the switching to alarm mode is necessary).

The results of the experiment are presented in Figures 6.29 and 6.30. We

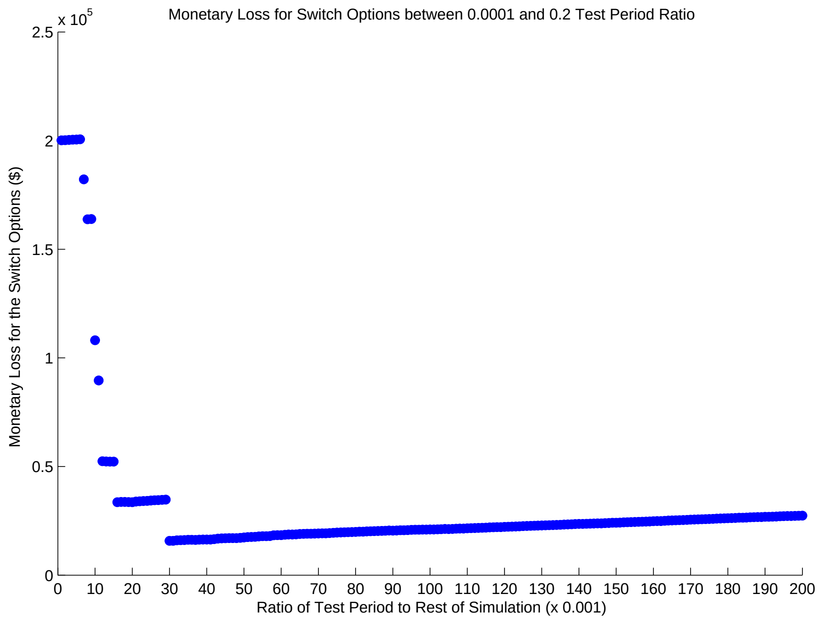


Figure 6.30: Monetary Loss for Test Period Lengths up to 20% of the Rest of Simulation Period

varied the length of the test period to see how it affects the cost of lost valuables (i.e. being less-than-necessary cautious to switch to alarm mode) and the payment to the security firm (i.e. being too cautious). We ran 10 cases of 100,000 minutes for each test period length. Figure 6.29 shows that initially losses are very high because the learning of necessary indicator values (i.e. sound values that require a switch) has not been completed yet. However, once the test period length reaches about 3% of the total simulation time (as shown in Figure 6.30 that is the close-up of Figure 6.29), it again starts to increase with the length of test period. This is because the security firm is paid $1 for each minute of the test period and this cost is not recovered by the slightly better tuned indicator values. Hence, there is an apparent trade-off between the costs of the test period and the gains received by learning when to switch. Furthermore, Figure 6.30 shows the *step* stair shaped cost curve and the sudden drops of losses happening after a sequence of the same monetary losses for different test period lengths. Those appear because certain lengths of the test period were necessary to learn important switching cases for the rest of the simulation.

Figures also show that staying always in the safe mode causes the loss of valuables ($200,000 average monetary loss), while keeping always the alarm mode on incurs the payment of $100,000 to the security firm. The switch options may result in higher losses than alarm always on strategy when the very short test periods causes the loss of valuables. However, with longer test periods, the switch option strategy yields losses very-close to optimal (lowest loss line in Figure 6.29 corresponding to the optimal switch decision, therefore unattainable in reality) results are achieved.

6.3.4 Conclusions and Future Work

In this section, we presented the application of switch options to dynamically compose complex services by optimally interconnecting basic services, and changing modes of operation in pervasive security applications. The work presented here addresses the issues of uncertainty arising in service activation and execution costs in a sensor network, as well as clever and efficient selection of operation modes for pervasive applications to reduce monetary costs.

Future work directions include improvement of our method via online detection of switch points, which is expected to decrease the costs incurred by running services simultaneously. Furthermore it would be interesting to implement the proposed method in a real testbed, and evaluate the advantages of our approach in a more realistic sensor or security application.

CHAPTER 7
Conclusions and Future Work

In this thesis, we have proposed using Probabilistic Context Free Grammars (PCFG) modeling for data produced during the operation of a network. We first presented our proposed algorithm for training a PCFG given the training data; this algorithm is more efficient than previously proposed ones. Then, we applied the algorithm to many domains, including event recognition in sensor networks, mobility modeling and mobility trace generation in mobile networks, social network behavior modeling, and service composition learning in service oriented sensor networks. We discussed for each of these applications the related work, and compared our methodology in terms of its representative power and complexity cost to other methods proposed in the literature. Based on these results, we conclude that PCFG modeling is a compact and efficient way of discovering and preserving data properties and using them later to regenerate the data, or provide classification ability in monitoring applications.

Independent of our PCFG-based modeling efforts, we also presented our contributions to metric-based service composition in sensor networks, as well as to application of switch options in pervasive sensor applications.

We have already discussed the future work for the subjects so far examined, in their respective sections separately. However, we would like to add here a more general discussion on future directions that can be based on our work.

The first interesting direction of future work is to further improve the PCFG construction method and to expand the applications of the PCFG modeling method to other domains, where appropriate. As aforementioned in Section 2, a pre-processing on the data to discover frequent (and meaningful) substrings, as well as the branching (applying multiple advantageous operations) during grammar construction are two directions we would like to pursue in the future to improve the grammar inference algorithm.

As a second direction of future work, we believe there are plenty of contribution opportunities for service computing in sensor networks. A new approach we have very recently started working on is policy-based service composition. Policies describe the rules which the service communications (e.g. which service can provide information to which others etc.) and composition schemes must adhere to, and clever design of service compositions compliant to the given policies is important for efficient processing of service requests. We aim to come up with methods that efficiently deal with distributed access to policy rules as well as their enforcement on service selections.

As a last direction of future work, we would like to further investigate market mechanisms, which is an emerging topic in networking applications. We would very much like to see the real world implementation of our current efforts in the switch options mechanism for pervasive applications.

LITERATURE CITED

[1] R. Agrawal, T. Imielinski, and A. N. Swami, “Mining association rules between sets of items in large databases,” in *Proc. ACM SIGMOD Int. Conf. Management of Data*, 1993, pp. 207−216.

[2] J. Han, H. Cheng, D. Xin, and X. Yan, “Frequent pattern mining: current status and future directions,” *Data Mining and Knowl. Discovery*, vol. 15, no. 1, pp. 55−86, Aug. 2007.

[3] B. Goethals, “Survey on frequent pattern mining,” Helsinki Inst. for Inf. Technol., Helsinki, Finland, Tech. Rep., 2003.

[4] R. Agrawal and R. Srikant, “Fast algorithms for mining association rules in large databases,” in *Proc. Int. Conf. Very Large Data Bases*, 1994, pp. 487−499.

[5] M. J. Zaki, “Scalable algorithms for association mining,” *IEEE Trans. Knowl. Data Eng.*, vol. 12, pp. 372−390, May 2000.

[6] J. Han, J. Pei, Y. Yin, and R. Mao, “Mining frequent patterns without candidate generation: A frequent-pattern tree approach,” *Data Mining and Knowl. Discovery*, vol. 8, no. 1, pp. 53−87, Jan. 2004.

[7] M. J. Zaki, “SPADE: An efficient algorithm for mining frequent sequences,” *Mach. Learning J.*, vol. 42, no. 1, pp. 31−60, Jan. 2001.

[8] V. Guralnik and G. Karypis, “Parallel tree-projection-based sequence mining algorithms,” *Parallel Comput. J.*, vol. 30, no. 4, pp. 443−472, Apr. 2004.

[9] E. Ukkonen, “On-line construction of suffix trees,” *Algorithmica*, vol. 14, no. 3, pp. 249−260, Sep. 1995.

[10] M. Eirinaki and M. Vazirgiannis, “Web mining for web personalization,” *ACM Trans. Internet Technol.*, vol. 3, no. 1, pp. 1−27, Feb. 2003.

[11] M. Spiliopoulou, “Web usage mining for web site evaluation,” *Commun. ACM*, vol. 43, no. 8, pp. 127−134, Aug. 2000.

[12] M. J. Zaki, C. D. Carothers, and B. K. Szymanski, “VOGUE: A variable order hidden Markov model with duration based on frequent sequence mining,” *ACM Trans. Knowl. Discovery from Data*, vol. 4, no. 1, pp. 1−31, Jan. 2010.

[13] A. Srivastava, S. Sural, and A. K. Majumdar, “Database intrusion detection using weighted sequence mining,” *J. Comput.*, vol. 1, no. 4, pp. 8−17, Jul. 2006.

[14] L. E. Baum, T. Petrie, G. Soules, and N. Weiss., “A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains,” *Ann. Math. Stat.*, vol. 41, no. 1, pp. 164−171, Feb. 1970.

[15] A. J. Viterbi, “Error bounds for convolutional codes and an asymptotically optimum decoding algorithm,” *IEEE Trans. Inf. Theory*, vol. IT-13, pp. 260269, Apr. 1967.

[16] C. D. Manning and H. Schutze, *Foundations of Statistical Natural Language Processing.* Cambridge, MA: MIT Press, 1999, pp. 381−407.

[17] A. Stolcke, “Bayesian learning of probabilistic language models,” Ph.D. dissertation, Dept. Comput. Sci., Univ. California, Berkeley, 1994.

[18] L. R. Rabiner, “A tutorial on hidden Markov models and selected applications in speech recognition,” *Proc. IEEE*, vol. 77, no.2, pp. 257−286, Feb. 1989.

[19] E. Charniak, C. Hendrickson, N. Jacobson, and M. Perkowitz, “Equations for part-of-speech tagging,” in *Proc. Nat. Conf. Artificial Intelligence*, 1993, pp. 784−789.

[20] T. E. Starner and A. Pentland, “Visual recognition of American sign language using hidden Markov models,” in *Proc. Int. Workshop Automatic Face and Gesture Recognition*, 1995, pp. 189−194.

[21] L. Song, U. Deshpande, U. C. Kozat, D. Kotz, and R. Jain, “Predictability of WLAN mobility and its effects on bandwidth provisioning,” in *Proc. IEEE INFOCOM*, 2006, pp. 1−13.

[22] A. J. Nicholson and B. D. Noble, “BreadCrumbs: forecasting mobile connectivity,” in *Proc. ACM Annu. Int. Conf. Mobile Computing and Networking*, 2008, pp. 46−57.

[23] A. Krogh, M. Brown, I. S. Mian, K. Sjolander, and D. Haussler, “Hidden Markov models in computational biology: Applications to protein modeling,” *J. Molecular Biol.*, vol. 235, no. 5, pp. 1501−1531, Feb. 1994.

[24] S. R. Eddy, “Profile hidden Markov models,” *Bioinformatics*, vol. 14, no. 9, pp. 755−763, Oct. 1998.

[25] D. M. Blei, A. Y. Ng, and M. I. Jordan, “Latent Dirichlet allocation,” *J. Mach. Learning Research*, vol. 3, no.7, pp. 993−1022, Mar. 2003.

[26] T. Griffiths and M. Steyvers, “A probabilistic approach to semantic representation,” in *Proc. Annu. Conf. Cognitive Science Society*, 2002, pp. 381−386.

[27] T. L. Griffiths and M. Steyvers, “Finding scientific topics,” in *Proc. Nat. Academy of Sciences 101*, 2004, pp. 5228−5235.

[28] T. P. Minka and J. Lafferty, “Expectation-propagation for the generative aspect model,” in *Proc. Uncertainty in Artificial Intelligence*, 2002, pp. 352−359.

[29] L. Cao and L. Fei-Fei, “Spatially coherent Latent Topic Model for concurrent segmentation and classification of objects and scenes,” in *Proc. IEEE Int. Conf. Computer Vision*, 2007, pp. 1−8.

[30] K. Farrahi and D. Gatica-Perez, “Mining human location-routines using a multi-level approach to topic modeling,” in *Proc. IEEE Int. Conf. Social Computing*, 2010, pp. 446−451.

[31] Y. Wang and G. Mori, “Human action recognition by semi-latent topic models,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, pp. 1762−1774, Oct. 2009.

[32] D. Xing and M. Girolami, “Employing latent Dirichlet allocation for fraud detection in telecommunications,” *Pattern Recognition Lett.*, vol. 28, no. 13, pp. 1727−1734, Oct. 2007.

[33] I. Biro, J. Szabo, and A. Benczur, “Latent dirichlet allocation in web spam filtering,” in *Proc. Int. Workshop Adversarial Information Retrieval on the Web*, 2008, pp. 21−24.

[34] A. McCallum, A. Corrada-Emmanuel, and X. Wang, “Topic and role discovery in social networks,” in *Proc. Int. Joint Conf. Artificial Intelligence*, 2005, pp. 786−791.

[35] K. Farrahi and D. Gatica-Perez, “What did you do today?: Discovering daily routines from large-scale mobile data,” in *Proc. ACM Int. Conf. Multimedia*, 2008, pp. 849−852.

[36] E. Linstead, C. Lopes, and P. Baldi, “An application of Latent Dirichlet Allocation to analyzing software evolution,” in *Proc. IEEE Int. Conf. Machine Learning and Applications*, 2008, pp. 813−818.

[37] D. Jurafsky et al., “Using a stochastic context-free grammar as a language model for speech recognition,” in *Proc. Int. Conf. Acoustics, Speech, and Signal Processing*, 1995, pp. 189−192.

[38] B. Roark, “Probabilistic top-down parsing and language modeling,” *Comput. Linguistics*, vol. 27, no. 2, pp. 249–276, Jun. 2001.

[39] Y. Sakakibara et al., “Stochastic context-free grammars for tRNA modeling,” *Nucleic Acids Research*, vol. 22, no. 23, pp. 5112–5120, Nov. 1994.

[40] Y. Sakakibara, “Grammatical inference in bioinformatics,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, pp. 1051–1062, Jul. 2005.

[41] D. Lymberopoulos, A. S. Ogale, A. Savvides, and Y. Aloimonos, “A sensory grammar for inferring behaviors in sensor networks,” in *Proc. Int. Conf. Information Processing in Sensor Networks*, 2006, pp. 251–259.

[42] T. Teixeira, E. Culurciello, J. H. Park, D. Lymberopoulos, and A. Savvides, “Address-event imagers for sensor networks: evaluation and modeling,” in *Proc. Int. Conf. Information Processing in Sensor Networks*, 2006, pp. 458–466.

[43] T. Teixeira, D. Lymberopoulos, E. Culurciello, Y. Aloimonos, and A. Savvides, “A lightweight camera sensor network operating on symbolic information,” presented at the Int. Workshop Distributed Smart Cameras, Boulder, CO, 2006.

[44] D. Lymberopoulos, A. Barton-Sweeney, T. Teixeira, and A. Savvides, “An easy to program sensor system for parsing human activities,” Yale Univ., New Haven, CT, Tech. Rep. 090601, 2006.

[45] H. Mitomi, F. Fujiwara, M. Yamamoto, and S. Taisuke, “Bayesian classification of a human custom based on stochastic context-free grammar,” *Syst. and Comput. in Japan*, vol. 38, no. 9, pp. 52–62, Aug. 2007.

[46] Y. A. Ivanov and A. F Bobick, “Recognition of visual activities and interactions by stochastic parsing,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, pp. 852–872, Aug. 2000.

[47] J. Earley, “An efficient context-free parsing algorithm,” *Commun. ACM*, vol. 13, no. 2, pp. 94–102, Feb. 1970.

[48] D. Moore and I. Essa, “Recognizing multitasked activities from video using stochastic context-free grammar,” in *Proc. AAAI–02*, 2002, pp. 770–776.

[49] K. M. Kitani, Y. Sato, and A. Sugimoto, “Deleted interpolation using a hierarchical Bayesian grammar network for recognizing human activity,” in *Proc. Int. Conf. Computer Communications and Networks*, 2005, pp. 239–246.

[50] E. Kiciman and A. Fox, "Detecting application-level failures in component-based internet services," *IEEE Trans. Neural Netw.*, vol. 16, pp. 1027−1041, Sep. 2005.

[51] S. C. Geyik and B. K. Szymanski, "Event recognition in sensor networks by means of grammatical inference," in *Proc. IEEE INFOCOM*, 2009, pp. 900−908.

[52] K. Lari and S. J. Young, "The estimation of stochastic context-free grammars using the inside-outside algorithm," *Comput. Speech and Language*, vol. 4, no. 1, pp. 35−56, Jan. 1990.

[53] J. Baker, "Trainable grammars for speech recognition," in *Proc. Spring Conf. Acoustical Society of America*, 1979, pp. 547−550.

[54] A. Stolcke and S. Omohundro, "Inducing probabilistic grammars by Bayesian model merging," in *Proc. Int. Colloq. Grammatical Inference and Applications*, 1994, pp. 106−118.

[55] S. F. Chen, "Building probabilistic models for natural language," Ph.D. dissertation, Dept. Comput. Sci., Harvard Univ., Cambridge, MA, 1996.

[56] G. S. Brodal, R. B. Lyngso, A. Ostlin, and C. N. S. Pedersen, "Solving the string statistics problem in time O(n log n)," in *Proc. ICALP*, 2002, pp. 728−739.

[57] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks," *IEEE Commun. Mag.*, vol. 40, no. 8 , pp. 102−114 , Aug. 2002.

[58] O. Amft, C. Lombriser, T. Stiefmeier, and G. Troster, "Recognition of user activity sequences using distributed event detection," in *Proc. European Conf. Smart Sensing and Context*, 2007, pp. 126−141.

[59] M. Li, Y. Liu, and L. Chen, "Nonthreshold-based event detection for 3D environment monitoring in sensor networks," *IEEE Trans. Knowl. Data Eng.*, vol. 20, pp. 1699−1711, Dec. 2008.

[60] S. Hongeng, R. Nevatia, and F. Bremond, "Video-based event recognition: Activity representation and probabilistic recognition methods," *Comput. Vision and Image Understanding*, vol. 96, no. 2, pp. 129−162, Nov. 2004.

[61] M. Marin-Perianu, C. Lombriser, O. Amft, P. Havinga, and G. Troster, "Distributed activity recognition with fuzzy-enabled wireless sensor networks," in *Proc. IEEE Int. Conf. Distributed Computing in Sensor Systems*, 2008, pp. 296−313.

[62] S. D. Tran and L. S. Davis, “Event modeling and recognition using Markov logic networks,” in *Proc. European Conf. Computer Vision*, 2008, pp. 610−623.

[63] M. Richardson and P. Domingos, “Markov logic networks,” *Mach. Learning*, vol. 62, no. 1, pp. 107−136, Feb. 2006.

[64] A. Y. Yang, R. Jafari, S. S. Sastry, and R. Bajcsy, “Distributed recognition of human actions using wearable motion sensor networks,” *J. Ambient Intell. and Smart Environments*, vol. 1, no. 2, pp. 103−115, Apr. 2009.

[65] J. Yamato, J. Ohya, and K. Ishii, “Recognizing human action in time-sequential images using hidden Markov model,” in *Proc. CVPR*, 1992, pp. 379−385.

[66] M. Brand, “The ”Inverse Hollywood Problem”: From video to scripts and storyboards via causal analysis,” in *Proc. AAAI/IAAI*, 1997, pp. 132−137.

[67] N. M. Oliver, B. Rosario, and A. P. Pentland, “A Bayesian computer vision system for modeling human interactions,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, pp. 831−843, Aug. 2000.

[68] M. Brand, N. Oliver, and A. Pentland, “Coupled hidden Markov models for complex action recognition,” in *Proc. CVPR*, 1997, pp. 994−999.

[69] S. C. Geyik, E. Bulut, and B. K. Szymanski, “PCFG based synthetic mobility trace generation,” in *Proc. IEEE Globecom*, 2010, pp. 1−5.

[70] D. B. Johnson, D. A. Maltz, and J. Broch, “The dynamic source routing protocol for multihop wireless ad hoc networks,” in *Ad Hoc Networking.* C. E. Perkins, Ed. Boston, MA: Addison-Wesley, 2001, pp. 139−172.

[71] European Telecommun. Standards Inst., “Selection procedures for the choice of radio transmission technologies of the UMTS (UMTS 30.03 version 3.2.0),” European Telecommun. Standards Inst., Sophia Antipolis, France, Tech. Rep. 101 112 V3.2.0, 1998.

[72] S. Ahmed, G. C. Karmakar, and J. Kamruzzaman, “An environment-aware mobility model for wireless ad hoc network,” *Elsevier Comput. Netw.*, vol. 54, no. 9, pp. 1470−1489, May 2010.

[73] L. Harfouche, S. Boumerdassi, and E. Renault, “Towards a social mobility model,” in *Proc. IEEE PIMRC*, 2009, pp. 2876−2880.

[74] L. Hu and L. Dittmann, “Heterogeneous community-based mobility model for human opportunistic network,” in *Proc. IEEE WiMob*, 2009, pp. 465−470.

[75] R. Calegari, M. Musolesi, F. Raimondi, and C. Mascolo, “CTG: A connectivity trace generator for testing the performance of opportunistic mobile systems,” in *Proc. ESEC/FSE*, 2007, pp. 415−424.

[76] N. Frangiadakis, M. Kyriakakos, S. Hadjiefthymiades, and L. Merakos, “Realistic mobility pattern generator: design and application in path prediction algorithm evaluation,” in *Proc. IEEE Int. Symp. Personal, Indoor and Mobile Radio Communications*, 2002, pp. 765−769.

[77] F. K. Karnadi, Z. H. Mo, and K. Lan, “Rapid generation of realistic mobility models for VANET,” in *Proc. IEEE Wireless Communications and Networking Conf.*, 2007, pp. 2506−2511.

[78] D. S. Tan, S. Zhou, J. Ho, J. Mehta, and H. Tanabe, “Design and evaluation of an individually simulated mobility model in wireless ad hoc networks,” in *Proc. Communication Networks and Distributed Systems Modeling and Simulation Conf.*, 2002, pp. 1−8.

[79] Y. Chang and H. Liao, “EMM: An event-driven mobility model for generating movements of large numbers of mobile nodes,” *Simulation Modelling Practice and Theory*, vol. 13, no. 4, pp. 335−355, Jun. 2005.

[80] D. Bhattacharjee, A. Rao, C. Shah, M. Shah, and A. Helmy, “Empirical modeling of campus-wide pedestrian mobility observations on the USC campus,” in *Proc. Vehicular Technology Conf.*, 2004, pp. 2887−2891.

[81] I. Stepanov, J. Hahner, C. Becker, J. Tian, and K. Rothermel, “A meta-model and framework for user mobility in mobile networks,” in *Proc. IEEE Int. Conf. Networks*, 2003, pp. 231−238.

[82] M. Kim, D. Kotz, and S. Kim, “Extracting a mobility model from real user traces,” in *Proc. IEEE INFOCOM*, 2006, pp. 1−13.

[83] C. Tuduce and T. Gross, “A mobility model based on WLAN traces and its validation,” in *Proc. IEEE INFOCOM*, 2005, pp. 664−674.

[84] W. Hsu, T. Spyropoulos, K. Psounis, and A. Helmy, “Modeling time-variant user mobility in wireless mobile networks,” in *Proc. IEEE INFOCOM*, 2007, pp. 758−766.

[85] K. Maeda, A. Uchiyama, T. Umedu, H. Yamaguchi, K. Yasumoto, and T. Higashino, “Urban pedestrian mobility for mobile wireless network simulation,” *Ad Hoc Netw.*, vol. 7, no. 1, pp. 153−170, Jan. 2009.

[86] S. Gunasekaran and N. Nagarajan, “An improved realistic group mobility model for MANET based on unified relationship matrix,” in *Proc. IEEE IACC*, 2009, pp. 1270−1274.

[87] J. Koberstein, H. Peters, and N. Luttenberger, “Graph-based mobility model for urban areas fueled with real world datasets,” in *Proc. Simutools Conf.*, 2008, pp. 1−8.

[88] D. Ashbrook and T. Starner, “Using GPS to learn significant locations and predict movement across multiple users,” *J. Pers. and Ubiquitous Comput.*, vol. 7, no. 5, pp. 275−286, Oct. 2003.

[89] L. Song and D. Kotz, “Evaluating location predictors with extensive wi-fi mobility data,” in *Proc. IEEE INFOCOM*, 2004, pp. 1414−1424.

[90] P. Jacquet, W. Szpankowski, and I. Apostol, “A universal predictor based on pattern matching,” *IEEE Trans. Inf. Theory*, vol. 48, pp. 1462−1472, Jun. 2002.

[91] J. Cleary and W. Teahan, “Unbounded length contexts for PPM,” *Comput. J.*, vol. 40, no. 2−3, pp. 67−75, Mar. 1997.

[92] A. Bhattacharya and S. K. Das, “LeZi-update: An information-theoretic approach to track mobile users in PCS networks,” *ACM/Kluwer Wireless Netw.*, vol. 8, no. 2−3, pp. 121−135, Mar. 2002.

[93] K. Gopalratnam and D. J. Cook, “Online sequential prediction via incremental parsing: The active LeZi algorithm,” *IEEE Intell. Syst.*, vol. 22, no. 1, pp. 52−58, Jan. 2007.

[94] M. Sipser, *Introduction to the Theory of Computation.* Boston, MA: PWS Publishing, 2006, pp. 115−123.

[95] X. Zhang, J. Kurose, B. N. Levine, D. Towsley, and H. Zhang, “Study of a bus-based disruption tolerant network: Mobility modeling and impact on routing,” in *Proc. ACM Annu. Int. Conf. Mobile Computing and Networking*, 2007, pp. 195−206.

[96] M. Piorkowski, N. Sarafijanovoc-Djukic, and M. Grossglauser, “A parsimonious model of mobile partitioned networks with clustering,” in *Proc. Int. Conf. Communication Systems and Networks*, 2009, pp. 1−10.

[97] S. Wasserman and K. Faust, *Social Network Analysis: Methods and Applications.* New York, NY: Cambridge Univ Press, 1994.

[98] S. C. Geyik, J. Xie, and B. Szymanski, “Behavior modeling with probabilistic context free grammars,” in *Proc. Int. Conf. Information Fusion*, 2010, pp. 1−8.

[99] F. Pianesi, M. Zancanaro, E. Not, C. Leonardi, V. Falcon, and B. Lepri, “Multimodal support to group dynamics,” *Pers. and Ubiquitous Comput.*, vol. 12, no. 3, pp. 181−195, Jan. 2008.

[100] R. Stiefelhagen, X. Chen, and J. Yang, “Capturing interactions in Meetings with omnidirectional cameras,” *Int. J. Distance Edu. Technol.*, vol. 3, no. 3, pp. 34−47, Jul. 2005.

[101] K. Otsuka, H. Sawada, and J. Yamato, “Automatic inference of cross-modal nonverbal interactions in multiparty conversations,” in *Proc. ICMI*, 2007, pp. 255−262.

[102] D. Hillard, M. Ostendorf, and E. Shriberg, “Detection of agreement vs. disagreement in meetings: Training with unlabeled data,” in *Proc. HLT−NAACL*, 2003, pp. 34−36.

[103] A. Pentland and A. Madan, “Perception of social interest,” presented at the IEEE Int. Conf. Computer Vision, Workshop Modeling People and Human Interaction, Beijing, China, 2005.

[104] I. McCowan, D. Gatica-Perez, S. Bengio, G. Lathoud, M. Barnard, and D. Zhang, “Automatic analysis of multimodal group actions in meetings,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, pp. 305−317, Mar. 2005.

[105] D. Zhang, D. Gatica-Perez, S. Bengio, and I. McCowan, “Modeling individual and group actions in meetings with layered HMMs,” *IEEE Trans. Multimedia*, vol. 8, pp. 509−520, Jun. 2006.

[106] A. Dielmann and S. Renals, “Automatic meeting segmentation using dynamic Bayesian networks,” *IEEE Trans. Multimedia*, vol. 9, pp. 25−36, Jan. 2007.

[107] S. Banerjee and A. I. Rudnicky, “Using simple speech-based features to detect the state of a meeting and the roles of the meeting participants,” in *Proc. Int. Conf. Spoken Language Processing*, 2004, pp. 2189−2192.

[108] P. Dai, H. Di, L. Dong, L. Tao, and G. Xu, “Group interaction analysis in dynamic context,” *IEEE Trans. Syst. Man Cybern. B, Cybern.*, vol. 39, pp. 34−42, Feb. 2009.

[109] A. Vinciarelli, “Role recognition in broadcast news using Bernoulli distributions,” in *Proc. IEEE Int. Conf. Multimedia and Expo*, 2007, pp. 1551−1554.

[110] C. Weng, W. Chu, and J. Wu, “Movie analysis based on roles social network,” in *Proc. IEEE Int. Conf. Multimedia and Expo*, 2007, pp. 1403−1406.

[111] N. P. Garg, S. Favre, H. Salamin, D. Z. Hakkani-Tur, and A. Vinciarelli, “Role recognition for meeting participants: An approach based on lexical information and social network analysis,” in *Proc. ACM Multimedia*, 2008, pp. 693−696.

[112] M. Zancanaro, B. Lepri, and F. Pianesi, “Automatic detection of group functional roles in face to face interactions,” in *Proc. Int. Conf. Multimodal Interfaces*, 2006, pp. 28−34.

[113] W. Dong, B. Lepri, A. Cappelletti, A. Pentland, F. Pianesi, and M. Zancanaro, “Using the influence model to recognize functional roles in meetings,” in *Proc. Int. Conf. Multimodal Interfaces*, 2007, pp. 271−278.

[114] S. Basu, T. Choudhury, B. Clarkson, and A. Pentland, “Learning human interactions with the influence model,” MIT Media Lab., Cambridge, MA, Tech. Rep. #539, 2001.

[115] S. Basu, T. Choudhury, B. Clarkson, and A. Pentland, “Towards measuring human interactions in conversational settings,” presented at the IEEE CVPR Int. Workshop Cues in Communication, Kauai, HI, 2001.

[116] R. Rienks and D. Heylen, “Dominance detection in meetings using easily obtainable features,” in *Proc. Workshop Machine Learning for Multimodal Interaction*, 2006, pp. 78−86.

[117] R. Rienks, D. Zhang, D. Gatica-Perez, and W. Post, “Detection and application of influence rankings in small group meetings,” in *Proc. Int. Conf. Multimodal Interfaces*, 2006, pp. 257−264.

[118] D. O. Olguin, B. N. Waber, T. Kim, A. Mohan, K. Ara, and A. Pentland, “Sensible organizations: Technology and methodology for automatically measuring organizational behavior,” *IEEE Trans. Syst. Man Cybern. B, Cybern.*, vol. 39, pp. 43−55, Feb. 2009.

[119] S. C. Geyik, B. K. Szymanski, P. Zerfos, and D. Verma, “Dynamic composition of services in sensor networks,” in *Proc. IEEE Int. Conf. Services Computing*, 2010, pp. 242−249.

[120] S. Dustdar and W. Schreiner, “A survey of web services composition,” *J. Web and Grid Services*, vol. 1, no. 1, pp. 1−30, Aug. 2005.

[121] R. Hull and J. Su, “Tools for composite web services: A short overview,” *ACM SIGMOD Rec.*, vol. 34, no. 2, pp. 86−95, Jun. 2005.

[122] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani, “A framework for QoS-aware binding and re-binding of composite web services,” *J. Syst. and Softw.*, vol. 81, no. 10, pp. 1754−1769, Oct. 2008.

[123] D. Ardagna and B. Pernici, “Adaptive service composition in flexible processes,” *IEEE Trans. Softw. Eng.*, vol. 33, no. 6, pp. 369−384, Jun. 2007.

[124] M. Klusch, A. Gerber, and M. Schmidt, “Semantic web service composition planning with OWLS-Xplan,” in *Proc. AAAI Fall Symp. Semantic Web and Agents*, 2005, pp. 55−62.

[125] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani, “An approach for QoS-aware service composition based on genetic algorithms,” in *Proc. Genetic and Evolutionary Computation Conf.*, 2005, pp. 1069−1075.

[126] P. Bartalos, “Effective automatic dynamic semantic web service composition,” *Inf. Sci. and Technol. Bulletin ACM Slovakia*, vol. 3, no. 1, pp. 61−72, Mar. 2011.

[127] F. Lécué and N. Mehandjiev, “Seeking quality of web service composition in a semantic dimension,” *IEEE Trans. Knowl. Data Eng.*, vol. 23, pp. 942−959, Jun. 2011.

[128] L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, “Qos-aware middleware for web services composition,” *IEEE Trans. Softw. Eng.*, vol. 30, pp. 311−327, May 2004.

[129] Z. M. Mao, R. H. Katz, and E. A. Brewer, “Fault-tolerant, scalable, wide-area internet service composition,” Univ. California, Berkeley, Tech. Rep. UCB/CSD−01−1129, 2001.

[130] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler, “The nesC language: A holistic approach to networked embedded systems,” in *Proc. ACM SIGPLAN*, 2003, pp. 1−11.

[131] B. Greenstein, E. Kohler, and D. Estrin, “A sensor network application construction kit (SNACK),” in *Proc. ACM SenSys*, 2004, pp. 69−80.

[132] G. Mainland, G. Morrisett, and M. Welsh, “Flask: Staged functional programming for sensor networks,” in *Proc. ACM SIGPLAN*, 2008, pp. 335−346.

[133] R. Newton, S. Toledo, L. Girod, H. Balakrishnan, and S. Madden, “Wishbone: Profile-based partitioning for sensornet applications,” in *Proc. USENIX Symp. Networked Systems Design and Implementation*, 2009, pp. 395−408.

[134] K. Whitehouse, F. Zhao, and J. Liu, “Semantic streams: A framework for composable semantic interpretation of sensor data,” in *Proc. EWSN*, 2006, pp. 5−20.

[135] J. Bronsted, K. M. Hansen, and M. Ingstrup, “A survey of service composition mechanisms in ubiquitous computing,” in *Proc. UbiComp Workshop*, 2007, pp. 87−92.

[136] T. Andrews et al. (2003, May). Business process execution language for web services (BPEL). [Online]. Available: http://www.ibm.com/developerworks/library/specification/ws-bpel/.

[137] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. (2001, Mar.). Web services description language (WSDL). [Online]. Available: http://www.w3.org/TR/wsdl.

[138] D. Martin et al. (2003, Nov.). OWL-S: Semantic markup for web services. [Online]. Available: http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/.

[139] D. Box et al. (2007, Apr.). Simple object access protocol (SOAP). [Online]. Available: http://www.w3.org/TR/soap/.

[140] J. Ibbotson, S. Chapman, and B.K. Szymanski, "The case for an agile SOA," presented at the Annu. Conf. ITA, Adelphi, MD, 2007.

[141] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*. R. E. Miller and J. W. Thatcher, Eds. New York, NY: Plenum Press, 1972, pp. 85−103.

[142] U. Feige, "A threshold of ln n for approximating set cover," *J. ACM*, vol. 45, no. 4, pp. 634−652, Jul. 1998.

[143] M. P. Papazoglou and W. van den Heuvel, "Service oriented architectures: Approaches, technologies and research issues," *VLDB J.*, vol. 16, no. 3, pp. 389−415, Jul. 2007.

[144] R. Sugihara and R. K. Gupta, "Programming models for sensor networks: A survey," *ACM Trans. Sensor Netw.*, vol. 4, no. 2, pp. 1−29, Mar. 2008.

[145] L. Mottola and G. P. Picco, "Programming wireless sensor networks: Fundamental concepts and state of the art," *ACM Comput. Surveys*, vol. 43, no. 3, pp. 1−51, Apr. 2011.

[146] J. Cardoso, A. P. Sheth, J. A. Miller, J. Arnold, and K. Kochut, "Quality of service for workflows and web service processes," *J. Web Semantics*, vol. 1, no. 3, pp. 281−308, Apr. 2004.

[147] S. Kona, A. Bansal, and G. Gupta, "Automatic composition of semantic web services," in *Proc. IEEE Int. Conf. Web Services*, 2007, pp. 150−158.

[148] A. V. Riabov, E. Bouillet, M. D. Feblowitz, Z. Liu, and A. Ranganathan, "Wishful search: Interactive composition of data mashups," in *Proc. ACM Int. Conf. World Wide Web*, 2008, pp. 775−784.

[149] E. Sirin, B. Parsia, D. Wu, J. Hendler, and D. Nau, “HTN planning for web service composition using SHOP2,” *Web Semantics: Sci., Services and Agents the World Wide Web*, vol. 1, no. 4, pp. 377−396, Oct. 2004.

[150] E. Sirin, B. Parsia, and J. Hendler, “Filtering and selecting semantic web services with interactive composition techniques,” *IEEE Intell. Syst.*, vol. 19, no. 4, pp. 42−49, Jul. 2004.

[151] E. Sirin, J. Hendler, and B. Parsia, “Semi-automatic composition of web services using semantic descriptions,” in *Proc. Web Services: Modeling, Architecture and Infrastructure Workshop in ICEIS*, 2003, pp. 17−24.

[152] D. van Thanh and I. Jorstad, “A service-oriented architecture framework for mobile services,” in *Proc. IEEE Telecommunications*, 2005, pp. 65−70.

[153] W. Tan, Y. Fan, M. Zhou, and Z. Tian, “Data-driven service composition in enterprise SOA solutions: A Petri Net approach,” *IEEE Trans. Autom. Sci. Eng.*, vol. 7, pp. 686−694, Jul. 2010.

[154] X. Wang, J. Wang, Z. Zheng, Y. Xu, and M. Yang, “Service composition in service-oriented wireless sensor networks with persistent queries,” in *Proc. CCNC*, 2009, pp. 1−5.

[155] J. W. Branch et al., “Towards middleware components for distributed actuator coordination,” presented at the Workshop Embedded Networked Sensors, Boston, MA, 2006.

[156] A. Bamis, N. Singh, and A. Savvides, “An architecture for dynamic reconfiguration of data flows in sensor networks,” Yale Univ., New Haven, CT, Tech. Rep., 2007.

[157] L. Luo, T. F. Abdelzaher, T. He, and J. A. Stankovic, “EnviroSuite: An environmentally immersive programming framework for sensor networks,” *ACM Trans. Embedded Comput. Syst.*, vol. 5, no. 3, pp. 543−576, Aug. 2006.

[158] A. Bakshi, V. K. Prasanna, J. Reich, and D. Larner, “The abstract task graph: A methodology for architecture-independent programming of networked sensor systems,” in *Proc. Workshop End-to-end, Sense-and-Respond Systems, Applications and Services*, 2005, pp. 19−24.

[159] W. Heinzelman, A. Murphy, H. Carvalho, and M. Perillo, “Middleware to support sensor network applications,” *IEEE Network*, vol. 18, no. 1, pp. 6−14, Jan. 2004.

[160] J. Wright et al., “A model-driven approach to the construction, composition and analysis of services on sensor networks,” presented at the Annu. Conf. ITA, London, UK, 2010.

[161] L. M. Silva, “Comparing error detection techniques for web applications: An experimental study,” in *Proc. IEEE Int. Symp. Network Computing and Applications*, 2008, pp. 144−151.

[162] L. J. Fulop et al., “Survey on complex event processing and predictive analytics,” Univ. Szeged, Szeged, Hungary, Tech. Rep., 2010.

[163] F. Salfner, M. Lenk, and M. Malek, “A survey of online failure prediction methods,” *ACM Comput. Surveys*, vol. 42, no. 3, pp. 1−42, Mar. 2010.

[164] Z. Zhang et al., “Gingko: Correlating causal paths in distributed systems,” in *Proc. IFIP Int. Conf. Network and Parallel Computing - Workshops*, 2007, pp. 762−767.

[165] M. Chen, A. Zheng, J. Lloyd, M. Jordan, and E. Brewer, “Failure diagnosis using decision trees,” in *Proc. Int. Conf. Autonomic Computing*, 2004, pp. 36−43.

[166] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann Publishers, 1993.

[167] L. Wu, L. Cheng, X. Qiu, and Y. Qiao, “A statistical approach to detect application-level failures in internet services,” in *Proc. IEEE Int. Conf. Fuzzy Systems and Knowledge Discovery*, 2009, pp. 155−159.

[168] G. Jiang, H. Chen, C. Ungureanu, and K. Yoshihira, “Multiresolution abnormal trace detection using varied-length n-grams and automata,” *IEEE Trans. Syst. Man Cybern. C, Appl. Rev.*, vol. 37, pp. 86−97, Jan. 2007.

[169] P. Bodik et al., “Combining visualization and statistical analysis to improve operator confidence and efficiency for failure detection and localization,” in *Proc. Int. Conf. Autonomic Computing*, 2005, pp. 89−100.

[170] M. P. Kasick, R. Gandhi, and P. Narasimhan, “Behavior-based problem localization for parallel file systems,” in *Proc. HotDep*, 2010, pp. 1−13.

[171] E. Hirota et al., “Multilayer failure detection method for network services based on distributed components,” in *Proc. IEEE/IFIP Network Operations and Management Symp.*, 2010, pp. 365−372.

[172] N. Ramanathan, K. Chang, R. Kapur, L. Girod, E. Kohler, and D. Estrin, “Sympathy for the sensor network debugger,” in *Proc. Int. Conf. Embedded Networked Sensor Systems*, 2005, pp. 255−267.

[173] B. K. Szymanski, S. Y. Shah, S. Geyik, S. Das, M. Chhabra, and P. Zerfos, “Market mechanisms for value of information driven resource allocation in sensor networks,” in *Proc. IEEE Int. Percom Workshop Information Quality and Quality Service for Pervasive Computing*, 2011, pp. 62−67.

[174] S. Geyik, B. Szymanski, P. Zerfos, and A. Mowshowitz, “Sensor service selection through switch options,” in *Proc. IEEE Int. Conf. Service Computing*, 2011, pp. 717−724.

[175] S. C. Geyik, S. Y. Shah, B. K. Szymanski, S. Das, and P. Zerfos, “Market mechanisms for resource allocation in pervasive sensor applications,” *Elsevier Pervasive Mobile Comput. J.*, vol. 8, no. 3, pp. 346−357, Jun. 2012.

[176] S. Geyik, E. Bulut, and B. Szymanski, “Utilizing PCFGs for modeling and learning service compositions in sensor networks,” in *Proc. IEEE Int. Conf. Service Computing*, 2012 (In Print).

[177] E. S. Schwartz and L. Trigeorgis, *Real Options and Investment under Uncertainty: Classical Readings and Recent Contributions.* Cambridge, MA: MIT Press, 2001, pp. 179−198.

[178] A. Mowshowitz, *Virtual Organization: Toward a Theory of Social Transformation Stimulated by Information Technology.* Westport, CT: Quorum Books, 2002, pp. 23−84.

[179] N. Kulatilaka, “Valuing the flexibility of flexible manufacturing systems,” *IEEE Trans. Eng. Manag.*, vol. 35, pp. 250−257, Nov. 1988.

[180] J. M. Harrison and J. A. Van Mieghem, “Multi-resource investment strategies: Operational hedging under demand uncertainty,” *European J. Operational Research*, vol. 113, no. 1, pp. 17−29, Feb. 1999.

[181] A. Mowshowitz, “On the market value of information commodities: III. Demand price,” *J. Amer. Soc. for Inf. Sci.*, vol. 43, no. 3, pp. 242−248, Apr. 1992.

[182] J. Ibbotson et al., “Sensors as a service oriented architecture: Middleware for sensor networks,” in *Proc. IEEE Int. Conf. Intelligent Environments*, 2010, pp. 209−214.

[183] S. Kalasapur, M. Kumar, and B. A. Shirazi, “Dynamic service composition in pervasive computing,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, pp. 907−918, Jul. 2007.

[184] E. Park and H. Shin, “Reconfigurable service composition and categorization for power-aware mobile computing,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, pp. 1553−1564, Nov. 2008.

[185] T. Repantis, Y. Drougas, and V. Kalogeraki, “Adaptive component composition and load balancing for distributed stream processing applications,” *Peer-to-Peer Netw. and Appl.*, vol. 2, no. 1, pp. 60−74, Mar. 2009.

[186] M. Alrifai and T. Risse, "Combining global optimization with local selection for efficient QoS-aware service composition," in *Proc. ACM Int. Conf. World Wide Web*, 2009, pp. 881−890.