Structured Prediction for Multi-AUV Situation Understanding

Nikolaos Kofinas and Michail G. Lagoudakis
Telecommunication Systems Institute (TSI)
Technical University of Crete (TUC)
Chania, Crete 73100, Greece

NOPTILUS

Table of Contents

1 Objective

In the context of the *Noptilus* project, Situation Understanding (WP6) is defined as:

> Cognitive ability of inferring high-level descriptions and representations of the current state of the environment.

The objective of this work package is summarized as follows:

> Automatically detecting, recognizing, understanding and predicting static underwater features and patterns as well as highly dynamic phenomena and events taking place and influencing the underwater operations.

The project deliverable of this work package is being developed as a system that supports the following functions, aligned with the two major tasks within WP6:

On-Board On-Line function [**Task 6.1**] The on-board on-line system functions as an *event recognizer* that aims to discover high-level discrete phenomena within vast amounts of raw sensor data streams. The goal of the system is to provide (i) abstraction from raw data to facilitate decision making at higher levels, (ii) robustness against sensor noise and environmental changes in different scenarios, and (iii) compression of sensor information as a result of abstraction. It should be noted that the description of recognized events has a much shorter representation and, therefore, can be utilized to communicate the abstract state of the AUV in an efficient way over the limited bandwidth of acoustic communication modems. Thus, this system can enhance coordination within the multi-AUV team.

Off-Board Off-Line function [**Task 6.2**] The off-line system serves as a training process for generating automatically the configuration of the on-line system (grammatical structure, probabilities, events, etc.). It is intended to serve as the *learning* platform, which will utilize past mission logs, possibly annotated by humans, to automatically derive patterns associated with events of a certain type, such as normal and abnormal behavior. The goal is to off-line generate a robust event recognition scheme, capable of effective on-line recognition.

Apart from the main two functions described above, the system delivered can be utilized in other meaningful ways. An off-board on-line use could be the real-time monitoring of the mission at a high level of description. Any information on recognized events transmitted by the on-board on-line system (acoustically or through Wi-Fi communication) can be presented to human operators at land/boat stations in ways that enhance awareness about the mission and simplifies the supervision of individual vehicles or monitoring of the whole mission itself. The two lines of research corresponding to Tasks 6.1 and 6.2 will be combined and merged in the context of Task 6.3, whose aim is to integrate the outcomes of the first two tasks.

The present document, Deliverable 6.2, titled "*Structured Prediction for Multi-AUV Situation Understanding*", according to the Noptilus DoW, *provides the details of* *the NOPTILUS situation understanding design that employs the structured prediction methodology, as well as the results obtained through the application of the learning procedure using available human-operator data.* This deliverable describes the research results obtained within Task 6.2, where focus is placed on the effective learning of a complete grammar for the recognition system on the AUVs. Learning a complete grammar is a problem that falls under the umbrella of structured prediction, given the fact that a grammar consists of both structural (production rules) and numerical (probabilities) components. The learned grammars will be used for on-board on-line event recognition, as investigated within Task 6.1 and is documented in Deliverable 6.1.

2 Definitions

- $a, b, c, \ldots$ represent symbols
- Σ represents an alphabet, that is, a set of symbols
- a^* means that the symbol a can appear zero or more times
- a^+ means that the symbol a can appear one or more times
- a^n means that the symbol a appears exactly n times
- w is a word, that is, any sequence of symbols
- $|w|$ denotes the length of a word w
- w^R is the reverse word of a word w
- O is a corpus, that is a finite set of words
- F^* represents the set of all words with zero or more symbols from F
- Σ^* represents all the words over some alphabet Σ
- $\mathcal{L}$ represents a language, that is, any subset of Σ^*
- G is a grammar over some alphabet
- $|G|$ is the length of grammar, the number of symbols required to be represented
- Lower-case letters in a grammar represent terminal symbols
- Upper-case letters in a grammar represent non-terminal symbols

3 Task 6.2 Motivation

As robot technology finds applications in the real world (search and rescue missions, daily household tasks, elderly care assistants, autonomous warehousing, autonomous cars, etc.), huge amounts of data are generated during autonomous robot missions. These data contain mostly diverse sensor measurements at a very fine time resolution and can hardly be analyzed by hand. In such applications, it is often desirable to recognize high-level events that may have occurred during a mission either online (recognition in real time, during the mission) or offline (post-processing recognition, after the mission). *Event Recognition* in robot missions is important, because high-level events describe situations that cannot be recognized directly from reading a few sensor data, but rather situations that can be recognized only by "mining" series of timed observations and identifying complex patterns. Important events may vary from a minor mechanical failure to something more elaborate, e.g. that the robot has drifted due to some external disturbance that is difficult to notice. Such events may have a significant impact in the progress of the mission. High-level mission events may include detection of an environmental change, completion of an individual sub-task, a systematic drift in robot's motion, a mechanical malfunction of a robot in the team, etc.

Event recognition in robot missions currently relies on human expertise and time-consuming data annotation. The designer of a robot team is typically responsible to provide the appropriate event recognizers, which will monitor various sensor streams in order to recognize events. The most effort-consuming procedure for the robot designer is the creation of specific recognizer (and possible handlers) for each individual important event. As robots become more complex and start becoming a part of our complex environment, researchers need a way to capture events without relying on specific recognizers. Another problem arises when a new, unexpected event occurs during a missions, which was not even anticipated during the design. In this case, either the event goes unnoticed or the researchers must analyze a large volume of logged data, annotate them to find the event and then create the corresponding recognizer for it. For these reasons, there is need to find a way to create event recognizers for robot missions without significant human involvement. To this end, our goal is to come up with a learning algorithm, which will create event recognizers automatically using annotated data from various missions. This annotation must be kept at high level, because the detailed annotation of data is, also, very effort-consuming for the designer.

A modern method to recognize events, investigated also in the context of NOPTILUS within Task 6.1, is to employ *Probabilistic Context-Free Grammars* (PCFGs), which are formal models from theoretical computer science widely used in natural language processing. The key component of a PCFG is a set of syntactic rules, which define how valid symbol sequences (words) can be derived using a discrete alphabet of symbols. The formalism of PCFGs provides an intuitive way to define a formal language consisting of all the valid words that can be derived from the grammar. PCFGs can capture complex patterns in discrete sequences and thus, can be used to parse incoming sensor data streams in order to detect patterns that may signal the occurrence of some event of interest. Our experimentation in Task 6.1 with such methods on data from *Autonomous Underwater Vehicle* (AUV) missions indicated that interesting events, such as collisions with the sea bottom, can be recognized by parsing sequences of sensor data, such as depth and pitch measurements, using an intuitive hand-written PCFG. In addition, the structure of PCFGs allows the construction of hierarchical event recognizers that may recognize complex events on the basis of recognized simpler events using a hierarchy of PCFGs. An advantage of this approach is that the PCFG itself is a human-readable structure, which can provide a clear explanation and/or interpretation of when and why an event occurs or becomes recognized. The main disadvantage of this approach is that the creation of a correct PCFG for just one event take a lot of effort and time. Our goal in this research is to provide a learning method which will be capable of learning appropriate PCFGs for Event Recognition directly from lightly-annotated mission data. Such a method will be very useful in creating PCFGs automatically for a variety of events, without putting much effort on the design.

4 Task 6.2 Contribution

The grammars proposed in Task 6.1 were hand-crafted. While the manual specification and tuning of grammars served the purpose of the case study, it is clearly a non-viable approach in the long run. There are several sensor measurements streams on the vehicles, the range of all possible events that may have to be recognized can hardly be understood in advance, and the human involvement may result in a rather tedious and error-prone task.

Each AUV carries a variety of different sensors and each one of them produces a large amount of data every second. It is impossible to create different grammars to capture all possible events that can recognized through the sensors, because (a) we do not have a proper definition of what an event of interest is and (b) it is difficult to identify the sensors that must be used to enable recognition of any target event. To deal with these problems, we focus on methods for learning grammars automatically from large data collections (past mission logs).

This task introduces a generic procedure which can be used to automatically construct PCFGs which encode sensor data sequences that typically appear during normal robot operation using recorded logs from past missions. The resulting PCFGs can be used to recognize abnormal events in new missions evidenced by sensor data sequences which cannot be interpreted as normal. The proposed procedure consists of two parts: (a) the transformation of sensor streams into discrete sequences either to form a training corpus offline or to generate input for online parsing and (b) a *Grammatical Inference* algorithm in order to learn a compact PCFG consistent with a given training corpus. The first part uses quantization methods in order to transform a stream of sensor data into sequences of symbols (words) that collectively can be viewed as members of a formal language.

The learning part relies on a local search method over the space of possible grammars using chunk and merge operations that modify the structure and the derivation power of a grammar. The search method aims to find a compact grammar that also maximizes its posterior probability, in a Bayesian sense, with respect to a given training corpus. The learning objective is to strike a balance between compactness and generalization, taking into account the fact that the training corpus contains only words that belong to the target language (positive examples).

The proposed procedure is evaluated on a variety of domains ranging from data sets generated by typical context-free grammars to data sets generated from real robot missions (AUV navigation). More specifically, our approach is demonstrated on the following tests: (a) learning of known textbook context-free grammars, (b) learning a PCFG recognizer for synthetic (sinusoidal) data, and (d) learning a PCFG recognizer for data generated during a mission with a NOPTILUS AUV. The results indicate that our approach is capable of producing reliable PCFG-based event recognizers, which may yield some false positive signals, but in general succeed in capturing abnormalities.

5 Formal Grammars

5.1 Alphabets and Languages

An *alphabet* is a finite set of symbols. The most common and well-known alphabet is the Latin alphabet $\Sigma = \{a, b, \ldots, z\}$, but any finite set of different symbols can be named an alphabet, e.g. $\Sigma = \{a, 0, 2, \%, \#\}$.

Given an alphabet, we can construct words; a *word* w is simply a sequence of one or more symbols taken from the alphabet. Thus, the word "*corresponding*" (not including the quotes) is a word over the Latin alphabet and the word "020202a%" is a word over the second alphabet defined above. The *length* $|w|$ of a word w is the number of symbols it contains. The set of all the possible words that can be constructed from a given alphabet Σ is denoted by Σ^*.

Now we can define a *language* $\mathcal{L}$ over an alphabet Σ as any subset of Σ^*, that is, an arbitrary set of words from Σ^*. It must be noted that every word by itself can be a language. Some extreme language cases are: $\Sigma^*, \{a\}, \{b\}, \{\ \}$

Note that a language $\mathcal{L}$ can be finite or infinite. Finite languages can be specified by listing all the possible words, however this cannot be done for infinite languages, thus we are using the following scheme:

$$\mathcal{L} = \{w \in \Sigma^* : \text{word } w \text{ satisfies property } P\}$$

In the above scheme, w are all the (infinite) possible words that belong in the language and P is a set of properties that all these words must satisfy. A simple example of such a language specification is the following:

$$\mathcal{L}_{ex} = \left\{w \in \{a, b, c\}^* : w \text{ has an even number of } a\text{'s and } b\text{'s and only two } c\text{'s}\right\}$$

Some of the words that belong to this language $\mathcal{L}_{ex}$ are: "*acca*", "*abababcc*", "*cc*", etc.

5.2 Grammars

Any language can be represented compactly or even be defined (or generated) using a grammar. A grammar G is a formal structure characterized by a set of terminal symbols, which are the base symbols forming sequences, a set of non-terminal symbols, which are intermediate symbols that characterize structural components of valid sequences, and a set of production rules. Formally, a grammar $G = (V, \Sigma, R, S)$ consists of the following:

- V is a finite set of non-terminal symbols
- Σ is an alphabet
- R is a finite set of production rules of the form $leftSide \rightarrow rightSide$
- S is the start symbol, a distinguished non-terminal symbol from V

A typical convention is to represent non-terminal symbols by uppercase letters and the start symbol by S, whenever this is possible. The length of a grammar $|G|$ is the total number of mathematical and alpharithmetic symbols required to be represented on paper.

It is possible to construct valid sequences (*words*) belonging to the language defined by a grammar through a process known as *derivation*. Derivation begins with the start symbol of the grammar and iteratively transforms the current symbol sequence of the derivation by applying rules of the grammar to the derivation, until the sequence contains only terminal symbols and no rule can be applied anymore. Depending on the exact form of the rules we have different types of grammars. The differentiation comes mostly on what is allowed in the left and the right sides of the grammar rules. These differentiations are described in detail below.

5.3 Context-Free Grammars

Context-Free Grammars (CFGs) are formal tools used widely in Computer Science and Natural Language Processing for specifying the syntax of programming and natural languages and for parsing documents to identify their syntactic correctness and structure. A CFG G is formally defined as a 4-tuple (V, Σ, R, S), where $R \subseteq V \times (\Sigma \cup V)^+$. This means that the left side of each rule is any single non-terminal symbol, whereas the right side is any non-empty sequence of terminal and non-terminal symbols.

The characterization *context-free* implies that rules can be applied to any non-terminal symbol in a derivation sequence, regardless of the context, that is, the symbols preceding or following the chosen symbol. The power of CFGs is that, given an arbitrary sequence of terminal symbols (a word), a *syntactic tree* can be constructed through a procedure known as *parsing* that describes the structural elements of the input word, provided that the word can be derived from the grammar. If the input word cannot be derived from the grammar, parsing yields no syntactic tree.

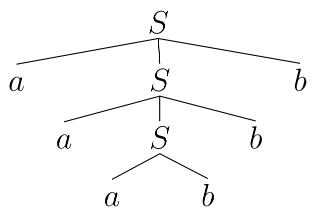
A simple context-free grammar with only two production rules that generates the language $\mathcal{L} = \{a^n b^n : n \geq 1\}$ is the following:

$$
\begin{aligned}
G &= (V, \Sigma, R, S) \\
V &= \{S\} \\
\Sigma &= \{a, b\} \\
R &= \{S \rightarrow aSb,\ S \rightarrow ab\}
\end{aligned}
$$

A common representation convention is to rewrite the rules R so that production rules referring to the same non-terminal symbol are grouped into a single rule with several productions. For example R in the grammar above can be written as $R = \{S \rightarrow aSc|ab\}$, where the special symbol | is used to separate different productions of the same rule. This convention is useful offers a compact representation. The word *aaabbb* can be derived as follows

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaabbb$$

and its syntactic tree is the following



In contrast, the word *aabbbb* cannot be derived from this grammar and thus, no syntactic tree exists in this case.

5.4 Probabilistic Context-Free Grammars

Probabilistic Context-Free Grammars (PCFGs) extend CFGs by adding a probability value to each production rule so that the probability values over all rules for a certain non-terminal symbol form a valid probability distribution (they are non-negative and they sum up to one). During derivation, a specific production of a rule for replacing a non-terminal symbol is drawn from the corresponding probability distribution. A PCFG implicitly defines a joint probability distribution over all possible sequences derived from the grammar. This probability value for any given sequence denotes "*how probable it is for the grammar to generate the given sequence*" and, therefore, how well the grammar *fits* the sequence and vise versa. Sequences that cannot be derived from the grammar have a derivation probability value of zero.

A formal definition of a PCFG G can be given as a 5-tuple:

$$G = (V, \Sigma, R, P, S)$$

It can be seen that the only difference compared to CFGs is the addition of P, which lists the *production probabilities* of each production rule $r \in R$. These probabilities reflect the likeliness of selecting some rules to substitute a non-terminal symbol during a derivation.

To make the representation of a PCFG simpler, the representation of P and R are represented jointly. The reason for this convention is that it makes easier to understand the probability that corresponds to a production of a rule. The example CFG of the previous subsection extended to PCFG will look like:

$$\begin{aligned}
G &= (V, \Sigma, R, S, P) \\
V &= \{S\} \\
\Sigma &= \{a, b\} \\
RP &= \{S \to aSb\ (0.2),\ S \to ab\ (0.8)\}
\end{aligned}$$

5.4.1 From a CFG to a PCFG

To transform a context-free grammar to a probabilistic one, we must assign probabilities to its production rules. To this end, a referenced corpus O, that is, a collection of words which can be derived from the grammar, is required. There are several learning algorithms whose goal is to infer probabilities that maximize the likelihood of PCFG with respect to the reference corpus O.

The most known among them is the Inside-Outside algorithm [1], which belongs to the family of Expectation-Maximization (EM) algorithms [2].

Inside-Outside takes into account all possible derivations that can produce the same word in the corpus and yields very accurate probabilities. This algorithm's complexity is $\mathcal{O}(n^3)$ where n is the number of production rules of the grammar.

5.4.2 Chomsky Normal Form

A $(P)CFG$ is said to be in *Chomsky Normal Form* (CNF) [3], when it follows these rules:

1. The right side of any production rule with the start symbol S on the left side consists of exactly **two** non-terminal symbols
2. The right side of any production rule with any other non-terminal symbol on the left consists of exactly **two** non-terminal symbols or **one** terminal symbol

As a consequence, a (P)CFG in CNF can produce words whose lengths are greater or equal to two. The simple PCFG that was described above in CNF becomes:

$$G = (V, \Sigma, R, S, P)$$
$$V = \{S, N_1, A, B\}$$
$$\Sigma = \{a, b\}$$
$$RP = \{S \to AN_1\ (0.2),\ S \to AB\ (0.8),\ N_1 \to SB\ (1.0),\ A \to a\ (1.0),\ B \to b\ (1.0)\}$$

The drawback of CNF is that the grammar becomes larger and harder to understand, but, on the other hand, as we shall see in the next section, the best parser available for parsing (P)CFGs assumes that the grammar is given in CNF. However, it is proven [4] that any (P)CFG can be easily transformed into an equivalent one in CNF, excluding words of length less than two.

5.5 Parsing

A parser is used every time it is needed to determine whether a word can be derived from a given grammar. There are a number of different formal ways to automatically construct a parser given a grammar [5]. One of them is the bottom-up approach, which is used by the `bison` parser generator. The problem with those parsers is that they are sacrifice generality for the sake of efficiency.

Since our grammars are PCFGs, there are two families of generic parsers, efficient enough to be considered: Earley [6] and Cocke-Younger-Kasami (CYK) [7] parsers. Typically, Earley parsers are considered to be superior to CYK parsers, since for a given input of length n and grammar of length $|G|$, they exhibit an $\mathcal{O}(n \cdot |G|)$ complexity. However, for ambiguous grammars, Earley parsers present the same complexity as CYK parsers, $\Theta(n^3 \cdot |G|)$. Additionally, CYK parsers are far easier to construct and use in practice.

5.5.1 CYK for CFGs

The CYK parser, shown in Algorithm 1, can only be used with context-free grammars in CNF. For any input word it asserts if this word can be derived from the grammar or not. The importance of the CYK algorithm stems from its high efficiency in certain situations. The worst case running time of CYK is $\Theta(n^3 \cdot |G|)$, where n is the length of the input word and $|G|$ is the size of the CNF context-free grammar G. This makes it one of the most efficient parsing algorithms in terms of worst-case asymptotic complexity, although other algorithms exist with better average running time in many practical scenarios.

5.5.2 CYK for PCFGs

If we were to parse a word with respect to a PCFG, it would be better to know whether it can be generated from the grammar and, also, the probability that this

Algorithm 1 CYK algorithm for CFG

Input: CNF CFG $G = (V, \Sigma, R, S)$ and word $= s_1 \cdots s_n$ where $n \geq 2$
Output: word $\in \mathcal{L}(G)$ or word $\notin \mathcal{L}(G)$

$N[i,j] := \emptyset$, $i = 1, \ldots, n$, $j = i, \ldots, n$
for $i := 1$ **to** n **do**
  $N[i,1] := \{A : (A \to s_i) \in R\}$
end for
for $j := 1$ **to** $n-1$ **do**
  for $i := 1$ **to** $n-j$ **do**
    for $k := i$ **to** $i+j-1$ **do**
      if $(A \to BC) \in R$ **and** $B \in N[i,k]$ **and** $C \in N[k+1, i+j]$ **then**
        $N[i, i+j] := N[i, i+j] \cup \{A\}$
      end if
    end for
  end for
end for
if $S \in N[1,n]$ **then**
  return word $\in \mathcal{L}(G)$
else
  return word $\notin \mathcal{L}(G)$
end if

can be done. The CYK algorithm can be modified to handle PCFGs with only some minor changes. The basic change is that table N in each cell stores additionally a probability value for each member of the corresponding set in that cell. Therefore, when we add a new entry (a left side of some production rule) to a cell of table N, we also calculate the corresponding probability. This probability takes into account the probabilities of all the components of the production rule, as shown in the complete description in Algorithm 2. If there is an update to an existing entry of the table, the highest probability is kept.

Figure 1 presents the resulting table N after a run of the CYK Algorithm 2. The input to this run is the PCFG presented above (Section 5.4) and the word: "*aaaabbbb*". The output of the resulting probability for this word is 0.0064.

5.6 Grammatical Inference

Grammatical inference [8] in machine learning is the process of learning a formal grammar from a set training set (a set of words, or a set of syntactic trees, or ..). More generally, grammatical inference is the way of learning new production rules that can parse (or generate) words that are part of the training set. There are various types of inference algorithms, each one of them focusing on languages that belong to different classes of languages. The problem of learning formal grammars for regular languages has been considered solved, since efficient algorithms have been proposed. On the other hand, there are no generic algorithms that can efficiently learn and

Algorithm 2 CYK algorithm for PCFG

Input: CNF PCFG $G = (V, \Sigma, R, P, S)$ and word $= s_1 \cdots s_n$ where $n \geq 2$
Output: $P(word)$ (if word $\in \mathcal{L}(G)$ then $P(word) > 0$, else $P(word) = 0$)

$N[i,j] := \emptyset$, $i = 1, \ldots, n$, $j = i, \ldots, n$
for $i := 1$ **to** n **do**
  $N[i,1] := \{A,\ P(A \to s_i) : (A \to s_i) \in R\}$
end for
for $j := 1$ **to** $n-1$ **do**
  for $i := 1$ **to** $n-j$ **do**
    for $k := i$ **to** $i+j-1$ **do**
      if $(A \to BC) \in R$ **and** $B \in N[i,k]$ **and** $C \in N[k+1, i+j]$ **then**
        $prob := P(A \to BC) * P_{N[i,k]}(B) * P_{N[k+1,i+j]}(C)$
        if $A \in N[i, i+j]$ **and** $P_{N[i,i+j]}(A) < prob$ **then**
          $P_{N[i,i+j]}(A) := prob$
        else if $A \notin N[i, i+j]$ **then**
          $N[i, i+j] := N[i, i+j] \cup \{A, prob\}$
        end if
      end if
    end for
  end for
end for
if $S \in N[1,n]$ **then**
  return $P_{N[1,n]}(S)$
else
  return 0
end if

handle languages with richer formalism, such as context-free and context-sensitive languages.

The Grammatical Inference algorithms can be categorized based on the type of the input training set:

1. **positive** data, where all input data belong to the target language
2. **positive** and **negative** data, where the input data are labeled as belonging or not to the target language

Additionally, some algorithms need an **oracle** that can answer membership queries. A membership query posted to the oracle, takes as input a word and answers if that word belongs to the target language or not. While this can be done easily, if the oracle knows the grammar that generates the target language, it is impossible to automatically answer such a query, if the grammar is unknown, which is common in real-world problems. Sometimes a human can play the role of the oracle, typically in natural language problems, but in most cases this cannot be done.

The data that the algorithms need as input can be categorized in:

a	A							S (0.0064)
a	A					S (0.032)	N_1 (0.032)	
a	A			S (0.16)	N_1 (0.16)			
a	A	S (0.8)	N_1 (0.8)					
b	B							
b	B							
b	B							
b	B							

Figure 1: CYK completed table

1. **words** which must be annotated as positive or negative
2. **syntactic trees** of pre-parsed positive words

As it can be understood, input data in the form of syntactic trees rely on heavy annotation and they are mainly used in natural-language processing, where such annotation exists.

It can be proven that for any given set of words generated by a grammar capable of infinite recursion, there is an indefinite number of grammars that could have produced the same data. Note that these grammars may in general generate different languages. E. Mark Gold proved [9] that it is impossible to learn a grammars for an infinite languages with certainty. More specifically, he showed that any formal language generated by a grammar capable of infinite recursion is un-learnable from positive words alone, in the sense that it is impossible to formulate a procedure that will discover with certainty the correct grammar given any arbitrary sequence of positive words.

6 Related Work

The idea of Grammatical Inference of context-free grammars has been successfully applied to a variety of fields, such as pattern recognition, computational biology, and natural language processing [10]. This is an indication that Grammatical Inference of CFGs is a promising direction towards our goal namely Event Recognition. The following subsections present the related work in Grammatical Inference of CFGs, discussing the advantages and disadvantages of each approach, as well as its limited exploitation so far in Event Recognition.

6.1 Problem-specific algorithms

This section reviews Grammatical Inference algorithms, which do not handle the whole class of context-free grammars, but only specific sub-classes of them.

Non-Terminal Separated Grammars Omphalos [11] was a context-free language learning competition, whose winner was Alexander Clark of King's College London. His winning algorithm [12] is limited to the class of Non-Terminal Separated (NTS) languages which is subclass of context-free languages. This approach is not suitable for our problem, as it cannot be determined in advance whether our training data (words) can be described as a language of the NTS class.

Contextual Binary Feature Grammars Clark et al. [13] defined a new class of grammars known as the Contextual Binary Feature Grammars (CBFG) and proposed an algorithm that can learn any language of this class. Since it has been proven that this class does not enclose all possible CFGs [14], the algorithm is not suitable for our problem. Additionally, their approach requires an oracle that can answer the membership queries, but in our problem there is no way to create such an oracle.

Substitutable Languages Clark et al. [15] defined another class of grammars, called Substitutable Grammars. Once again, this class does not include all possible context-free grammars and, thus, it is not suitable for our approach. It should be mentioned that this approach is one of the few that guarantee polynomial-time complexity.

6.2 Generic algorithms

This section reviews Grammatical Inference algorithms, which handle the whole class of context-free grammars.

Binary Feature Grammars Clark et al. [16] describe an algorithm that can learn any language belonging to the class of Binary Feature Grammars (BFG) in polynomial time. The class of context-free languages is a subclass of BFGs and, thus, this approach seems to be suitable. The downside of this approach is that it needs an oracle that can answer the membership queries which is not available in our problem.

String Kernels This approach [17] handles Grammatical Inference in a different way. Instead of learning the structure of a grammar, e.g. the non-terminal symbols and production rules, it learns how to represent a language as hyperplanes in a high-dimensional feature space. Unfortunately, this representation provides no insight in the structure of the language itself and appears is a black box to the end user. While this approach can handle all context-free languages, we did not use it because our goal is to infer the structure of the target language and thus understand the nature of the event.

Minimum Description Length There are several approaches [18, 19] that try to utilize the idea of minimum description length to learn any context-free grammar.

The idea of minimum description length describes a way to compare two grammars based on their representation length and on the effective compression they achieve on the initial training corpus. These algorithms are based on local search initiated with a naive grammar that lists all words of the training corpus as separate production rules. At each step, they create all the possible successors of the current grammar, using various grammar modification operations, and they select the best successor to continue with. This process is repeated until no improvement can be achieved in the neighborhood of the current grammar. These local search approaches are not efficient enough for our problem, because even for a simple grammar they require a large number of positive data and often yield an over-generalized grammar, which derives many words outside the target language.

Emile This algorithm [20] works with a corpus of only positive words. The basic idea of the algorithm is that it arranges all the symbols of all the words on both sides of a two dimensional matrix. Each cell (i, j) of the matrix is marked, if the sequence $s_i s_j$, where s_i is the symbol corresponding to row i and s_j is the symbol corresponding to column j, is part of any word in the corpus. Then, the algorithm tries to find clusters inside this matrix and creates non-terminals from those clusters. While this algorithm is interesting, it was tested in our problem and the results was not satisfying. It failed to generalize and it created grammars that were over-fitted to the training corpus.

Iterative Bi-clustering Tu et al. [21] presented an approach similar to Emile. The key difference is that this approach generates a PCFG directly in CNF. The first step of the algorithm is also to create a matrix similar to that of Emile, but the cells of the matrix store the number of times the sequence $s_i s_j$ appears in the corpus. Then, it uses a bi-clustering algorithm to extract the best bi-cluster from the matrix. At this point, a new rule is created based on the extracted sub-matrix. This procedure cannot generalize and, thus, the authors introduced a second procedure, which checks whether there are possible merges between two existing rules of the grammar. If there are any, then the corresponding two rules are merged into a single rule referring to a new non-terminal symbol. Finally, the rows and columns of the matrix are updated to match the new non-terminal symbol. These two procedures are repeated until the matrix is reduced to 1×1. This approach seemed to be very suitable for our problem, because it can learn any context-free grammar, as well as calculate the probabilities of the rules directly from the corpus. However, its main disadvantage is the CNF requirement, which prevents performing efficient local search. Figure 2 shows an example we obtained from a simple implementation while investigating this method. It can be easily verified that the learned grammar correctly represents the obvious symmetry in the strings derived by the grammar given as set of input positive strings. Here, we should also mention that no empty string as given as input, thus the grammar represents the language $L = \{a^n b^n : n > 1\}$.

$aabb$	$N_0 \rightarrow AB$
ab	$N_0 \rightarrow AN_1$
$aaaabbbb$	$N_1 \rightarrow N_0B$
$aaaaaabbbbbb$	$A \rightarrow a$
$aaaaaaabbbbbbb$	$B \rightarrow b$
$aaaaabbbbb$	$S \rightarrow N_0$

(a) Input positive strings (b) The learned CFG ($a^nb^n, n > 1$)

Figure 2: A simple grammatical inference example with positive data.

Bayesian Learning This local search approach [22] tries to find the best successor of a grammar by comparing the posterior probabilities of the successors grammars. The key idea of this approach is similar to that of Bi-clustering and Minimum Description Length that were discussed above. It uses the same operations as the Minimum Description Length, which are quite similar to the ones that Bi-clustering approach uses. The goal of this approach is to find a grammar that maximizes the posterior probability $P(G|O)$ of a grammar G over a training corpus O. This probability cannot be computed directly, but is given by Bayes' rule:

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

This algorithm is the most suitable for our problem and served as the base idea behind our learning algorithm.

6.3 Event Recognition with the use of (P)CFG

The idea of using Grammatical Inference to learn the appropriate recognizers for Event Recognition has also been investigated by Geyik et al. [23]. Their goal is to recognize three distinct events in video sequences from a parking lot: (a) Enter and Park, (b) Enter and Leave, and (c) Leave Parking. Their approach needs fully annotated data with these three events.

A similar approach [24] tried to categorize computer log files, by checking if they have an anomaly or not. In this work, the input data that used to train the CFG consisted of highly-annotated log files, where all useless data (not related to the target anomaly) had been eliminated.

While these two pieces of work tried to accomplish something similar to what we are trying to accomplish, they did not exploit effectively a principled learning approach of those used for Grammatical Inference in other domains. In both cases, certain heuristics were introduced for convenience, resulting in questionable learning outcomes (grammars).

7 Normal and Abnormal Events

In the case study of Task 6.1, we focused on collision events with the sea bottom, because of prior information provided by experts and the existence of logs with such events. Clearly, there is a large number of events that one may be interested in during an AUV mission, such as getting caught in a fishing net, colliding with a moving obstacle, facing a strong underwater current, entering new waters, etc. The design of recognizers for specific events is a time-consuming process. Additionally, the annotation of data to indicate where an event (if any) has occurred within a mission is also a difficult task and sometimes impossible to accomplish.

Our experience so far indicates that it is rather difficult to write down all possible events that may be useful to recognize, let alone the problem of finding the corresponding grammars or mission logs containing occurrences of these events. To avoid these difficulties, we suggest a different, somewhat generic, approach in which we try to identify significant events without going through time-consuming annotations for specific events. To this end, we decided to separate events into two broad categories: the first containing all normal events that occur during normal AUV operation, such as accelerating, submerging, emerging, turning, etc., and the second containing all other events which we can describe as abnormal. Abnormal events are rare events that typically do not occur during a normal mission.

The target of our approach now is to learn grammars that model those normal events. There is an abundance of data from normal mission which can be used towards this purpose. If these grammars are inferred successfully, any sequence of data that cannot be recognized as belonging to the language of the learned grammars can be considered as abnormal. When an abnormal sequence is detected, a signal may be send to the high-level behavior to signal the presence of a rare or abnormal event.

The advantage of this approach is that we only need data from normal missions to proceed. These can be easily collected simply by observing missions and by keeping only the ones that were completed without any problems or surprises. Additionally, the detection of abnormal events as anything that occurs outside the nominal measurements of normal missions may lead to detection of rare, but possibly interesting, events that cannot be easily characterized or localized to specific measurements in advance. On the other hand, the main disadvantage is where to draw the line between normal and abnormal data. If the normal data used for training do not cover the entire range of normal missions, it is possible that ignored normal events will be detected as abnormal. Also, if abnormal data are included (by mistake) in the training set, the corresponding abnormal events will never be recognized.

8 Data Logs to Words

Robot mission data logs typically consist of streams of different sensor measurements (accelerometer, gyrometer, depth meter, GPS, etc.). The first step towards learning a PCFG for Event Recognition is to translate robot mission data to words. In particular, we must specify an alphabet to represent the possibly continuous sensor measurements as discrete symbols. Thus, we need to map sensor values to symbols or in other words quantize the data. Then, the readings of a sensor over time can be translated to words by choosing appropriate time windows.

8.1 Quantization

Quantization is the procedure of constraining a continuous set of values to a relatively small discrete set [25]. In our approach this small set is our alphabet. The quantizer is responsible for this discretization and maps the input values to the corresponding symbols from the alphabet. The size of the desired alphabet is given by the user and determines the granularity of the discretization. As a convention any chosen alphabet is enhanced by two extra symbols to indicate possible measurements above and below the typical range of the sensor.

In our approach we tried two types of quantizers:

- The **scalar (uniform)** quantizer creates n uniformly distributed quantization regions over the range of the input values, where n is the size of our desired alphabet.
- The **Lloyd-max** quantizer also creates n quantization regions. The difference is that it distributes these regions over the range of the input values based on the minimization of the Mean-Squared Error between the discretized and the continuous data.

In practice, we observed that these two approaches produced similar results and thus in the rest of this work we will adopt the scalar quantizer which is more intuitive and easier to visualize.

8.2 Corpus creation

Once the alphabet and the translation to symbols have been determined, we must find a way to forms words from the stream of data. In their work Geyik et al. [23] had already words to work with, because their data logs had a known start and end point and therefore it was trivial to form the corresponding words. In our problem, after quantization we have to deal with large sequences of symbols, one such sequence for each sensor from each mission.

A naive approach is to take every such sequence as a single word. Focusing on one sensor, each mission contributes a large word in the corpus. The main disadvantage of this approach is the creation of very large words. The CYK parsing algorithm we are using can parse only complete words and thus, under this approach, we must wait for a mission to end and only then can we parse the resulting word to recognize an event. As a result, an event can only be recognized after the end of the mission. Another disadvantage is that we need a lot of missions to create a large enough corpus for training the grammar. Additionally, events of interest typically do not span the entire length of a mission, but rather they occur at specific times during a mission. Besides the difficulty of recognizing localized events into long words, even a successful event recognizer in this case would not be able to provide information about the time the event occurred during the mission.

Due to these disadvantages, we decided that it is more suitable to break the mission into many words to add to the corpus. The guiding principle in doing so, is to choose an appropriate time window over the length of the mission which is likely to include events of interest. This approach will create a large amount of words into the corpus from fewer missions. The disadvantage of this approach is that restricting the length of the words may degenerate the underlying language to be a member of the more restricted class of regular languages.

To split the sequence of data of a mission into smaller non-overlapping words, we investigate three different ways:

1. Words of a standard fixed length n.
2. Words whose length follows a uniform distribution over a fixed range $[n - \frac{n}{2} : n + \frac{n}{2}]$.
3. Words whose length follows a normal distribution $\mathcal{N}(n, \frac{n}{2})$ around a fixed length n.

At this point it must be mentioned that the user needs to provide three values as input: the size of the alphabet, the nominal length of words n, and the splitting method (standard, uniform, or normal).

8.3 Quantization and corpus creation example

In this section, we will present an example of the quantization and corpus creation method described above. We will use an alphabet of size 4, a nominal word length equal to 12, and a normal distribution to split the "mission" into words. Our data will be provided by a cosine function in the time window $[0 : 4\pi]$. Figure 3 presents the generated data and the uniform quantization regions. As we can see the 4 symbols of the alphabet are b, c, d, e enhanced with the extra symbols a, f which are reserved for outlier values outside a typical range.

The quantized log sequence translated to symbols is the following:

"b b b b b b b b b b b b b b b b b b b b b c c c c c c c c c c c d d d d d d d d
d d e e e e e e e e e e e e e e e e e e e e e e e e e e e e e e e e e e e e e e e e e e d
d d d d d d d d d d c c c c c c c c c c b b b b b b b b b b b b b b b b b b b b b
b b b b b b b b b b b b b b b b b b b b b c c c c c c c c c c c d d d d d d d d d
d e e e e e e e e e e e e e e e e e e e e e e e e e e e e e e e e e e e e e e e e e e e e d d
d d d d d d d d c c c c c c c c c c c b b b b b b b b b b b b b b b b b b b b b"

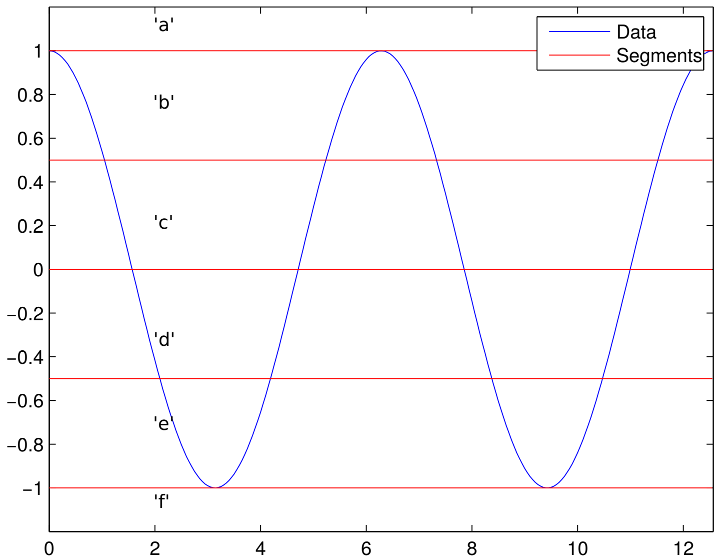


Figure 3: Data and quantization regions

The next step is to use a normal distribution $\mathcal{N}(12, 6)$ to split this large word to smaller ones. The resulting corpus has the following 20 words:

1. "b b b b b b b b b b b b b b"
2. "b b b b b b b b c c c c c c c c"
3. "c c c c d d d d d d d"
4. "d d d d e e e e e e e e"
5. "e e e e e e e e e e e e e e e e e e e e e e e e e"
6. "e e e e e e e e e e e d d d d d d d d d d d c c c c"
7. "c c c c c c c b b b b b b b b"
8. "b b b b b b b b b b b b b b b b b b b"
9. "b b b"
10. "b b b b b b b b b"
11. "b b b b b b b c c c c"
12. "c c c c c c c c d d d d d d d d"
13. "d d d"
14. "d e e e e e e e e e e e e e e e"
15. "e e e e e e"
16. "e e e e e e e e e e e e e"
17. "e e e e e e e e e e e d d d d d d"
18. "d d d d d c c c c c c c c c c"
19. "c c b b b b b b b b b b b b b b b b b"
20. "b b b b b"

9 Grammar Learning

During our research we found out that the Bayesian unsupervised learning approach is very suitable to our problem. The reason is that we do not know if our data belong to a specific subclass of context-free grammars and thus we need a generic approach that is able to learn any context-free grammar.

9.1 Learning Objective

The goal in our learning algorithm is to find the grammar G that maximizes the posterior probability over a training corpus O:

$$G^* = \arg\max_G P(G|O)$$

There is no known way to compute this probability and therefore, we use Bayes' rule:

$$G^* = \arg\max_G \frac{P(G)P(O|G)}{P(O)} = \arg\max_G P(G)P(O|G)$$

where $P(G)$ is the prior probability of the grammar and $P(O|G)$ is the probability of the current corpus given the grammar (likelihood).

Grammar prior There is not straight-forward way to define the prior probability of a grammar and, thus, we are adopted a choice which is based only on the description length of the grammar. This choice is the most commonly used in the literature [18, 19, 22], when the prior of a grammar must be calculated. The key idea is that the shortest grammar is more probable among all other grammars, based on Occam's Razor principle (simpler is better). We define that each symbol used in the representation of a grammar, increases its description length by one. Therefore, the prior probability of a grammar can be calculated as:

$$P(G) = \frac{1}{2^{|G|}}$$

where $|G|$ is length of the grammar. The exponential in the denominator penalizes heavily long representations. In other words, each addition of a symbol, reduces the prior by half. As we observed in our experiments, the base of the exponential has no significant impact, as long as it is larger than 1. To provide an example, consider the following grammar:

$$\begin{aligned} G &= (V, \Sigma, R, S) \\ V &= \{S\} \\ \Sigma &= \{a, b, c\} \\ R &= \{S \rightarrow abc \mid bbc\} \end{aligned}$$

To calculate the prior we take into account only the representation of the rules. The length of this representation is 9, because there is one symbol on the left side (S), six symbols in total on the right sides and two separator symbols ($\rightarrow$, $|$). Thus, the prior of the grammar is:

$$P(G_{prior}) = \frac{1}{2^9} = 0.001953$$

Likelihood The likelihood $P(O|G)$ is the probability of grammar G to produce the entire corpus O. Assuming that the words in the corpus O are iid, the likelihood can be expressed as the product of all parsing probabilities of the words in corpus O given grammar G:

$$P(O|G) = \prod_{w \in O} P(w|G)$$

where $P(w|G)$ is the probability of word w to be derived from grammar G.

9.2 Learning strategy

Since we can compute the prior $P(G)$ and the likelihood $P(O|G)$ of any grammar G given a corpus O, we now focus on how to setup a search procedure in order to find a grammar that maximizes the posterior $P(G|O)$.

Note that the search space of possible grammars is not a vector space, as common in maximum-likelihood optimization problems. The implication is that commonly used techniques, such as gradient following, cannot be applied. In addition, the search space is rather complex and cannot be generated systematically due to the diversity in the structure of grammars (non-terminal symbols, rules, productions, probabilities, start symbol). As a result, systematic search algorithms cannot be applied to cover the entire search space and therefore search completeness cannot be guaranteed by any search algorithm.

Additionally, we note that our search problem aims to find the best possible grammar in the search space. Even though we can evaluate the goodness of any grammar, any search algorithm applied to our problem can never be sure if the best possible grammar has been reached. The implication is that no termination criterion can be defined.

For these reasons we focus on local search algorithms, whereby search is initialized at some initial guess (a complete grammar) and proceeds iteratively by checking for an improved grammar within the "neighborhood" of the current guess and moving there. The neighborhood of a grammar can be defined in numerous different ways even though there is no single choice which is better than the others; our definition of a neighborhood will be based on two common grammar manipulation operations (chunk and merge) [19, 22]. The details of our local search procedure are described in the following sections.

9.3 Initialization of the PCFG

The initialization of the grammar for our local search procedure consists of three steps:

1. Create non-terminals to replace the terminal symbols in the corpus
2. Introduce a Start symbol with one production for each word of the corpus

3. Calculate the count (probability) for each distinct production

Step 1: From symbols to non-terminals In the first step we create non-terminals to replace all the terminal symbols in the training corpus. There is a one-to-one correspondence between the terminal symbols and the newly-created non-terminals. These non-terminals are created only to simplify our implementation and they will be ignored in all subsequent learning steps. To give an example, consider the following training corpus:

$$O = \left\{ \begin{array}{cccccccc} a & b & c & c & c & c & & \\ a & a & a & b & c & & & \\ a & a & b & b & b & c & & \\ a & a & b & b & c & c & & \\ a & a & a & b & c & c & c & c \\ a & a & a & a & a & a & b & c \\ a & a & b & b & b & c & & \end{array} \right\}$$

There are three unique terminal symbols that appeared in this training corpus a, b, and c and thus, three new non-terminal symbols N_1, N_2, and N_3 will be created. Each one of them will have single production $N_1 \rightarrow a$, $N_2 \rightarrow b$, and $N_3 \rightarrow c$. Now, all the occurrences of the terminal symbols in the corpus will be replaced with the corresponding non-terminal symbols and the training corpus becomes:

$$O_{new} = \left\{ \begin{array}{cccccccc} N_1 & N_2 & N_3 & N_3 & N_3 & N_3 & & \\ N_1 & N_1 & N_1 & N_2 & N_3 & & & \\ N_1 & N_1 & N_2 & N_2 & N_2 & N_3 & & \\ N_1 & N_1 & N_2 & N_2 & N_3 & N_3 & & \\ N_1 & N_1 & N_1 & N_2 & N_3 & N_3 & N_3 & N_3 \\ N_1 & N_1 & N_1 & N_1 & N_1 & N_1 & N_2 & N_3 \\ N_1 & N_1 & N_2 & N_2 & N_2 & N_3 & & \end{array} \right\}$$

The complete grammar, at this point, has four non-terminal symbols and three productions:

$$\begin{aligned} G_{init_1} &= (V, \Sigma, R, S) \\ V &= \{S, N_1, N_2, N_3\} \\ \Sigma &= \{a, b, c\} \\ R &= \{N_1 \rightarrow a,\ N_2 \rightarrow b,\ N_3 \rightarrow c\} \end{aligned}$$

Step 2: Production of the Start symbol Given that the corpus now contains only non-terminal symbols, a single rule for the start symbol S will be created with as many productions as the words in the corpus:

$$
\begin{aligned}
G_{init_2} &= (V, \Sigma, R, S) \\
V &= \{S, N_1, N_2, N_3\} \\
\Sigma &= \{a, b, c\} \\
R &= \left\{ \begin{array}{lllllllllll}
S & \rightarrow & N_1 & N_2 & N_3 & N_3 & N_3 & N_3 & & \\
S & \rightarrow & N_1 & N_1 & N_1 & N_2 & N_3 & & & \\
S & \rightarrow & N_1 & N_1 & N_2 & N_2 & N_2 & N_3 & & \\
S & \rightarrow & N_1 & N_1 & N_2 & N_2 & N_3 & N_3 & & \\
S & \rightarrow & N_1 & N_1 & N_1 & N_2 & N_3 & N_3 & N_3 & N_3 \\
S & \rightarrow & N_1 & N_1 & N_1 & N_1 & N_1 & N_1 & N_2 & N_3 \\
S & \rightarrow & N_1 & N_1 & N_2 & N_2 & N_2 & N_3 & & \\
N_1 & \rightarrow & a & & & & & & & \\
N_2 & \rightarrow & b & & & & & & & \\
N_3 & \rightarrow & c & & & & & & &
\end{array} \right\}
\end{aligned}
$$

This grammar, despite its large size, derives precisely the words of the corpus O, but it lacks in compactness. For a complete PCFG, the only missing part at this point are the production probabilities.

Step 3: Production counts In this final step, we assign un-normalized probabilities to all productions of the grammar. For simplicity, we refer to the un-normalized probabilities as the counts of the productions. First,we delete all duplicate productions and then we assign to them a count which reflects the number of occurrences in the corpus:

$$
\begin{aligned}
G_{init_3} &= (V, \Sigma, R, P, S) \\
V &= \{S, N_1, N_2, N_3\} \\
\Sigma &= \{a, b, c\} \\
RP &= \left\{ \begin{array}{llllllllll r}
S & \rightarrow & N_1 & N_2 & N_3 & N_3 & N_3 & N_3 & & & (1) \\
S & \rightarrow & N_1 & N_1 & N_1 & N_2 & N_3 & & & & (1) \\
S & \rightarrow & N_1 & N_1 & N_2 & N_2 & N_2 & N_3 & & & (2) \\
S & \rightarrow & N_1 & N_1 & N_2 & N_2 & N_3 & N_3 & & & (1) \\
S & \rightarrow & N_1 & N_1 & N_1 & N_2 & N_3 & N_3 & N_3 & N_3 & (1) \\
S & \rightarrow & N_1 & N_1 & N_1 & N_1 & N_1 & N_1 & N_2 & N_3 & (1) \\
N_1 & \rightarrow & a & & & & & & & & (19) \\
N_2 & \rightarrow & b & & & & & & & & (12) \\
N_3 & \rightarrow & c & & & & & & & & (14)
\end{array} \right\}
\end{aligned}
$$

This is the complete initial PCFG, which is nevertheless over-fitted to the training corpus and does not generalize beyond the training corpus. This generalization and the compactness of the grammar will be achieved through our learning algorithm.

Note that the prior $P(G_{init})$ of this grammar can be easily computed from the definition and it will be rather small due to the lack of compactness in the representation.

$$\begin{array}{lllllllllll} N_1 & \rightarrow & a & a & a & b & c & d & & (15) \\ N_2 & \rightarrow & a & b & c & a & a & d & & (22) \\ N_3 & \rightarrow & d & d & a & b & c & a & d & (9) \end{array} \Longrightarrow \begin{array}{llllllll} N_1 & \rightarrow & a & a & N_4 & d & & (15) \\ N_2 & \rightarrow & N_4 & a & a & d & & (22) \\ N_3 & \rightarrow & d & d & N_4 & a & d & (9) \\ N_4 & \rightarrow & a & b & c & & & (46) \end{array}$$

Figure 4: An example of the chunking operation

The structure of G_{init} allows us to compute the likelihood $P(O|G_{init})$ directly from the grammar, G_{init} since the corpus O has now been incorporated into the grammar:

$$P(O|G_{init}) = \prod_{r \in R_{init}} \prod_{p_r \in r} \left(\frac{P_{init}(p_r)}{\sum_{q_r \in r} P_{init}(q_r)} \right)^{P_{init}(p_r)}$$

where $P_{init}(p_r)$ is the probability of production p_r (or the corresponding count in case of un-normalized probabilities). The underlying principle is that each production p is used $P(p)$ times to parse the corpus and thus its probability must appear $P(p)$ times within the likelihood, as reflected in the exponent.

9.4 Grammar manipulation: Chunk and Merge

To create the neighborhood of the current grammar in the search space, we present two widely-used grammar manipulation operations (chunk and merge) which modify the grammar to create its successors.

Non-terminal chunking The chunk operator creates a new non-terminal symbol and the corresponding rule so that the new symbol derives a specific sub-sequence (of length at least two) that appears in the productions of the grammar. When a new rule is created through chunking, all the occurrences of this sub-sequence are replaced by the new non-terminal symbol. As a result, this operator can change the grammar length but does not modify the language of the grammar. A chunk that shrinks the grammar (reduces the length) will increase the prior and thus, the posterior will be increased too. This operator is incapable of affecting the generalization of the grammar, but it can produce a diversity of new non-terminals and rules which will be used in sub-sequence merging operations. Figure 4 shows a simple example of the chunking operator. In this scenario, the sub-sequence that will be chunked is "$a\ b\ c$", the new non-terminal symbol is N_4, and the corresponding new rule is $N_4 \rightarrow a\ b\ c$. As mentioned above, in all the productions of the grammar, the appearances of the sub-sequence "$a\ b\ c$" are replaced by the non-terminal N_4 and the new rule have been added to the grammar. Additionally, the total size of the grammar, in this case, is reduced by one, because:

1. 9 occurrences of symbols have been removed (three replacements of "$a\ b\ c$")

$$\begin{array}{lllllll} N_1 & \rightarrow & a & N_2 & a & N_3 & c \\ N_2 & \rightarrow & a & N_1 & a & N_3 & d \\ N_3 & \rightarrow & d & N_2 & d & c & d \end{array} \Longrightarrow \begin{array}{lllllll} N_1 & \rightarrow & a & N_2 & a & N_2 & c \\ N_2 & \rightarrow & a & N_1 & a & N_2 & d \\ & | & d & N_2 & d & c & d \end{array}$$

Figure 5: An example of the merging operation

2. 3 occurrences of the new symbol (N_4) were added through the replacements
3. 5 symbols were added to represent the new rule ($N_4 \rightarrow a\ b\ c$)

As mentioned before, after each chunk operation only the prior probability of the grammar changes. More specifically, the chunk operation adds/subtracts symbols in/from the representation of the grammar G_{old} and, thus, the posterior of the new grammar G_{new} after a chunk operation can be calculated by:

$$P(G_{new}|O) = P(G_{old}|O)\frac{1}{2^{|G_{new}|-|G_{old}|}}$$

The fraction on the right is known as the gain in the prior over the previous grammar:

$$PGain_{prior} = \frac{1}{2^{|G_{new}|-|G_{old}|}}$$

Non-terminal symbol merging The merge operator condenses two existing non-terminal symbols N_x and N_y into one by merging their production and eliminating one of the symbols (e.g. merge productions of N_y into N_x and delete N_y). All the occurrences of the eliminated symbol (N_y) in all other productions of the grammar are replaced by the symbol that remains (N_x). It must be noted that the non-terminal symbols that derive only terminal symbols do not participate in the merge operation. In Figure 5 a simple example of the merging operation is presented. In this scenario, the rules that will be merged are N_2 and N_3 and they are merged into N_2. Also, all occurrences of symbol N_3 in the grammar are replaced by N_2. The merge operation affects both the prior and the likelihood of the grammar, because it shrinks and generalizes the grammar. The length of the grammar is reduced at least by one, because of the deletion of at least one symbol. Additionally, the grammar may shrink even more because of the merging of identical productions that may appear after the replacement of the eliminated symbol. In any case, the prior of the grammar is increased. Additionally, the merge operation affects the generalization of the grammar, because after merging, all productions of the merged symbols are now available at all of their occurrences. Due to this generalization, the resulting grammar can potentially derive words outside the original corpus. The likelihood can be increased or decreased depending on the rules being merged and, thus, it must be recomputed after each merge. A naive way to accomplish that is to re-parse the whole initial corpus to update the derivation probability of each word. This procedure, however, is rather slow and will adds unnecessary overhead to the running time of the learning algorithm. Since the effects of the merge operation are in general localized, we can compute the likelihood incrementally considering the previous likelihood and the changes made by the operation:

$$P(O|G_{new}) = \prod_{r \in R_{new}} \prod_{p_r \in r} \left(\frac{P_{new}(p_r)}{\sum\limits_{q_r \in r} P_{new}(q_r)} \right)^{P_{new}(p_r)}$$

$$= P(O|G_{old}) \frac{\prod\limits_{r \in \widehat{R}_{new}} \prod\limits_{p_r \in r} \left(\frac{P_{new}(p_r)}{\sum\limits_{q_r \in r} P_{old}(q_r)} \right)^{P_{new}(p_r)}}{\prod\limits_{r \in \widehat{R}_{old}} \prod\limits_{p_r \in r} \left(\frac{P_{old}(p_r)}{\sum\limits_{q_r \in r} P_{old}(q_r)} \right)^{P_{old}(p_r)}}$$

where $\widehat{R}_{new}$ and $\widehat{R}_{old}$ are the subsets of rules R_{new} and R_{old} respectively affected by the merge operation (note that $\widehat{R}_{new}$ and $\widehat{R}_{old}$ are always different). Again, the fraction on the right is known as the gain in the likelihood over the previous grammar:

$$PGain_{likelihood} = \frac{\prod\limits_{r \in \widehat{R}_{new}} \prod\limits_{p_r \in r} \left(\frac{P_{new}(p_r)}{\sum\limits_{q_r \in r} P_{old}(q_r)} \right)^{P_{new}(p_r)}}{\prod\limits_{r \in \widehat{R}_{old}} \prod\limits_{p_r \in r} \left(\frac{P_{old}(p_r)}{\sum\limits_{q_r \in r} P_{old}(q_r)} \right)^{P_{old}(p_r)}}$$

The above incremental computation is feasible due to the structure of the initial grammar which incorporates the corpus O (Section 9.3). If the merge operation affects only the productions of the two merged symbols and, also, no pair of these productions collapses into one, as in the example of Figure 5, the likelihood decreases. To see this, notice that the probabilities of the two merged rules in $\widehat{R}_{old}$ will collapse and split over more choices (the merged productions) within one rule in $\widehat{R}_{new}$. On the other hand, if the merge operation causes the collapse of productions in any rule of the grammar, the likelihood may increase or decrease.

Independently of the operation applied, the total gain of the new grammar over the inital posterior is given by:

$$PGain_{new} = PGain_{prior}\ PGain_{likelihood}\ PGain_{old}$$

9.5 Effective posterior computations

Representation of the posterior During the first steps of experimentation with our implementation, we found out that the posterior took very small values (less than 10^{-50}) and we started to run into numerical problems due to the precision of our machine. In order to resolve these problems and make the implementation numerically stable, we store the logarithm of the posterior instead of the posterior itself, a common practice in such calculations based on the monotonic nature of the logarithmic function:

$$G^* = \arg\max_G P(G|O) = \arg\max_G \log P(G|O) = \arg\max_G \big(\log P(G) + \log P(O|G)\big)$$

where log is the base 10 logarithmic function. Similarly, we store the logarithm of the gain $Gain$ instead of the gain $PGain$ itself, which can now be updated as follows:

$$Gain_{new} = Gain_{prior} + Gain_{likelihood} + Gain_{old}$$

where $Gain_{prior}$ is now given by:

$$Gain_{prior} = -(|G_{new}| - |G_{old}|) \log 2$$

and $Gain_{likelihood}$ is now given by:

$$Gain_{likelihood} = P_{new}(p_r) \sum_{r \in \widehat{R}_{new}} \sum_{p_r \in r} \log \left(\frac{P_{new}(p_r)}{\sum\limits_{q_r \in r} P_{old}(q_r)} \right) - P_{old}(p_r) \sum_{r \in \widehat{R}_{old}} \sum_{p_r \in r} \log \left(\frac{P_{old}(p_r)}{\sum\limits_{q_r \in r} P_{old}(q_r)} \right)$$

Additionally, with this representation we transform the divisions into subtractions and the powers into multiplication which yield a significant speedup in the execution.

Normalization and regularization of the posterior During the implementation phase, we also observed that the two components of the posterior, the prior and the likelihood, do not take values in comparable ranges. This problem goes unnoticed under chunk operations, because chunks change only the prior and but not the likelihood. However, in a merge operation bot the prior and the likelihood, but the likelihood is the dominant component because of a high value range. We noticed that the problem was amplified, when we had large corpora because, as we mentioned above, the initial grammar contains un-normalized counts. An example that demonstrates the problem is the following: Consider the merge of two rules which have only one production each and both productions have a count equal to 100. We make the assumption that this merge does not change anything else in the grammar. If we merge these two rules, the grammar is shrank by only one symbol so the prior gain is:

$$Gain_{prior} = -\left((|G_{old}| - 1) - |G_{old}|\right) \log 2 = \log 2 = 0.3010$$

and the likelihood gain is:

$$Gain_{likelihood} = 2\left(100 \log\left(\frac{100}{200}\right) - 100 \log\left(\frac{100}{100}\right)\right) = -60.21$$

We can see that the two numbers differ by two orders of magnitude and in fact this was the reason that our first implementation was generating grammars that were over-fitted to the training corpus, simple because merge operations seemed to decrease the gain over the posterior. The solution to this problem is simple; when we construct the grammar, we divide all counts in the grammar with the total number of words in the corpus (corpus normalization). With this simple modification, the value of the gain of the likelihood after any merge operation is restricted to the range $[-1 : +1]$. While this solution seemed to work, we found out that now the algorithm tended to create over-generalized grammars. The reason, in this case, is that, if a merge shrinks the grammar by more than three symbols, the prior gain is equal to 0.9030, which is large enough to allow "bad" merges to be chosen. In other words, the dominant component now becomes the prior. We experimented with different ways of calculating the prior beyond the proposed one, but we did not overcome this problem. The solution we adopted is to introduce a non-negative regularization factor λ to the prior:

$$G^* = \arg\max_G \big(\lambda \log P(G) + \log P(O|G)\big)$$

This regularization factor tries to make the prior comparable to the likelihood and, as we show in Section 10, for small values of λ this solution effectively prevents over-generalization.

9.6 Search strategy

Maximum chunk size In describing the chunk operation, no limit was set for the maximum length of the sequence being chunked. It is possible to leave the operator unrestricted and search for chunks of any length. Nevertheless, the enumeration of all possible chunks of any length for a grammar with large productions (such as a just initialized grammars) is slow. For this reason, we defined the maximum subsequence length of any possible chunk equal to 15. This value can be easily changed in order to adapt our algorithm to different situations, where a bigger maximum chunk length may be useful. Moreover, we experimented with a maximum chunk length equal to two in order to force the learning algorithm to learn grammars directly in CNF format. This idea came from Tu et al. [21], where they proposed a similar chunking step (the bi-clustering step) with chunk length equal to two. Our results were not satisfiable, because local search under this constraint gets easily stuck to local maxima. A common trick to escape from local maxima is to use look-ahead techniques, which allow to discover good successors that lie beyond a few intermediate bad ones.

Best-First Search and Beam Search The first local search algorithm we implemented was *Best-First Search* (BFS). BFS is a greedy local search algorithm, which selects and proceeds to the best grammar among all successors, by comparing their overall gains. The basic implementation of the BFS does not use look-ahead and, thus, it easily gets stuck to local maxima. While the following context-free languages:

$$\mathcal{L}_1 = \{a^n b^n : n \geq 1\}$$
$$\mathcal{L}_2 = \{a^{2n} : n \geq 1\}$$
$$\mathcal{L}_3 = \{(ab)^n (cd)^n : n \geq 1\}$$

can be easily learned using BFS, BFS fails to learn more complex languages, such as palindromes:

$$\mathcal{L}_4 = \{ww^R : w \in \{a, b\}^*\}$$

or languages that describe correct mathematical expressions with addition and parentheses such as the following:

$$a + a + (a + a), \quad (((a + (a + a)))), \quad (a + a) + (a + a)$$

An extension of BFS is the *Beam Search* (BS) algorithm, which is also a greedy local search algorithm, but it can explore simultaneously more that one paths. BS has two parameters: *beam depth*, which is the maximum number of nodes (grammars) in the beam, and *beam width*, which is the maximum number of unexpanded nodes in the beam. The depth controls how many nodes the algorithm can use for look-ahead and the width how many different paths it can follow. BS starts by inserting into the beam the first node (initial grammar), which is marked as "unexpanded". At each iteration, it expands the unexpanded node, which yields the largest gain among all the unexpanded nodes in the beam. All of its successors are inserted into the beam, marked as "unexpanded", while the expanded node gets marked as "expanded" and is re-inserted into the beam. After that, the beam is checked to find out if the top $\#depth$ nodes contain at most $\#width$ unexpanded ones. If this upper limit is violated, the unexpanded node with the least gain value is deleted from the beam and the check is repeated, until there is no violation of the upper limit. Finally, all nodes that are not included in the top $\#depth$ nodes are deleted from the beam. The process is repeated until all top $\#depth$ nodes are marked as "expanded" and then the BS terminates by returning as goal node the one with the largest gain value among all the remaining nodes in the beam. A special case of BS is obtained, when we set depth and width equal to 1 then BS reduces to BFS. If we set width equal to 1 and depth equal to m, then BS becomes BFS with m-nodes look-ahead. While BS is a greedy, non-optimal, local search algorithm, it can give very good results and in most cases it discovers very good grammars. The width and the depth can be easily changed by the user to fit the problem at hand; in our case, we set width equal to 5 and depth equal to 17.

Recalculation of Probabilities After the search phase, the algorithm ends up with a PCFG whose probabilities may not model the corpus accurately any more. This may happen, because after the application of a merge operation, the counts may become invalid and they cannot be recalculated without re-parsing the whole corpus. As mentioned before, re-parsing of the entire corpus is a rather slow procedure and, therefore, we chose to skip it and let the algorithm continue with inaccurate counts.

We observed that in practice this has no significant impact on the construction of the grammar and, therefore, we chose to recalculate the correct probabilities only after search is completed and returns with good grammar. We recalculate the probabilities of the final grammar using the Inside-Outside algorithm [1]. During this phase, it is possible that some productions will end up with probabilities less than a small number ϵ. This small value indicates that such a production is either useless or only a few words in the corpus use it in their derivations. For this reason, we prune these productions from the grammar and then we re-parse the entire corpus. If all words are still parsed with probability greater than zero, then these productions remain pruned. Otherwise, they are added back to the grammar productions since they play a key role to the correct parsing of the corpus. In our implementation, we chose $\epsilon = 10^{-6}$.

Incremental Search While the algorithm presented so far works well, when the corpus does not contain a large number of distinct words, we noticed that it fails to find a correct grammar, when the corpus contains a large number of distinct words. An illustrative example can be given using the language $\mathcal{L} = \{a^n b^n : n \geq 1\}$. The PCFG for this language is the following:

$$\begin{aligned} G &= (V, \Sigma, R, S, P) \\ V &= \{S, A, B\} \\ \Sigma &= \{a, b\} \\ RP &= \{S \to ASB\ (p),\ S \to AB\ (1-p),\ A \to a\ (1.0),\ B \to b\ (1.0)\} \end{aligned}$$

When $p = 0.2$, generating a total of ten thousand words from this grammar results in a corpus, which has on average about six distinct words (the first six words up to $n = 6$). In this case, our search algorithm works fine and reconstructs the true grammar. However, when $p = 0.9$ generating a equally-sized corpus yields about 70 distinct words. Our search fails to reconstruct the original grammar and terminal with an over-generalized grammar, which represents the language $\mathcal{L} = \{a^n b^m : n, m \geq 1\}$. To overcome this problem we decided to break the initial corpus into batches of words in order to feed the learning algorithm with these batches, one at a time. This incremental search procedure consists of the following steps:

1. Copy the initial corpus to a primary corpus
2. Sort the words in the primary corpus from the most frequent to the less frequent
3. Form the first training batch using the first k distinct words
4. Initialize the first PCFG using the first training batch
5. Execute the search procedure until termination
6. Parse the entire primary corpus with the resulting PCFG
7. Transfer all parsed words from the primary corpus to a secondary corpus

8. Recalculate the probabilities of the PCFG using the secondary corpus
9. If the primary corpus is not empty, form a new training batch with the k most-frequent distinct words of the primary corpus, otherwise terminate
10. Use the new training batch to append the Start rule of the current grammar with productions that can derive the words of the new batch
11. Go to Step 5

This incremental search procedure can easily handle corpora containing a large number of distinct words with better control on over-generalization. The value of parameter k can be arbitrary; in our implementation we frequently set it to be equal to one-tenth of the total size of the initial corpus.

9.7 Learning example

We will present a simple example of our learning algorithm on the problem of learning the language $\mathcal{L}_{example} = \{a^n c b^n : \ n \geq 1\}$ from a small corpus $O_{example}$ with 18 words, which is listed bellow:

1. $a\ c\ b$
2. $a\ c\ b$
3. $a\ c\ b$
4. $a\ c\ b$
5. $a\ c\ b$
6. $a\ c\ b$
7. $a\ c\ b$
8. $a\ a\ c\ b\ b$
9. $a\ a\ c\ b\ b$
10. $a\ a\ c\ b\ b$
11. $a\ a\ c\ b\ b$
12. $a\ a\ c\ b\ b$
13. $a\ a\ c\ b\ b$
14. $a\ a\ a\ c\ b\ b\ b$
15. $a\ a\ a\ c\ b\ b\ b$
16. $a\ a\ a\ c\ b\ b\ b$
17. $a\ a\ a\ c\ b\ b\ b$
18. $a\ a\ a\ a\ c\ b\ b\ b\ b$

The corpus is chosen to be small in order to make the example easier to understand and, thus, the incremental search approach will not be used. Also, the width of the Beam Search will be equal to one and the depth equal to two (one-step look-ahead), thus at any time the beam will contain at most two grammars and at most one unexpanded. Finally, we will set the max chunk size equal to five in order to keep the example relatively small.

The initial grammar, according to Section 9.3, will be:

$$
\begin{aligned}
G_{example_1} &= (V, \Sigma, R, P, S) \\
V &= \{S, N_1, N_2, N_3\} \\
\Sigma &= \{a, b, c\} \\
RP &= \left\{ \begin{array}{lllllllllll}
S & \rightarrow & N_1 & N_2 & N_3 & & & & & & (0.39) \\
 & | & N_1 & N_1 & N_2 & N_3 & N_3 & & & & (0.33) \\
 & | & N_1 & N_1 & N_1 & N_2 & N_3 & N_3 & N_3 & & (0.22) \\
 & | & N_1 & N_1 & N_1 & N_1 & N_2 & N_3 & N_3 & N_3 \; N_3 & (0.06) \\
N_1 & \rightarrow & a & & & & & & & & (1.00) \\
N_2 & \rightarrow & c & & & & & & & & (1.00) \\
N_3 & \rightarrow & b & & & & & & & & (1.00)
\end{array} \right\}
\end{aligned}
$$

The algorithm begins by initializing the gain to zero, $Gain_1 = 0$, and inserting $G_{example_1}$ into the beam:

$$beam = \{G_{example_1}(Gain = 0, unexpanded), empty\}$$

The next step is to expand this grammar, since it is the only node in the beam. It marks this grammar as expanded, finds all its successors, and re-inserts them into the beam. Due to the size of the beam width being one, only one successor will be produced. At this step, only the chunk operator can be used, because we do not have two non-terminal symbols which can be merged. As already mentioned, N_1, N_2, and N_3 do not participate in the merge operation. All the possible chunks are the following:

1. N_1N_2 (0)
2. $N_1N_1N_2$ (1)
3. $N_1N_1N_1N_2$ (0)
4. $N_1N_1N_1N_1N_2$ (-2)
5. N_2N_3 (0)
6. $N_2N_3N_3$ (1)
7. $N_2N_3N_3N_3$ (0)
8. $N_2N_3N_3N_3N_3$ (-2)
9. N_1N_1 (0)
10. $N_1N_1N_1$ (-1)
11. $N_1N_1N_1N_1$ (-3)
12. N_1N_2 (0)
13. $N_2N_2N_2$ (-1)
14. $N_2N_2N_2N_2$ (-3)
15. $N_1N_2N_3$ (3)
16. $N_1N_1N_2N_3N_3$ (5)
17. $N_1N_1N_2N_3$ (3)
18. $N_1N_1N_1N_2N_3$ (1)
19. $N_1N_2N_3N_3$ (3)
20. $N_1N_2N_3N_3N_3$ (1)

The numbers in parentheses denote by how many symbols each chunk shrinks the grammar. As we can observe, chunk 16 shrinks the grammar the most (five symbols) and, thus, this is the one selected to produce the single successor. The new gain is:

$$Gain_2 = Gain_1 + 5 \log 2 = 0 + 1.5051 = 1.5051$$

The resulting grammar is:

$$
\begin{aligned}
G_{example_2} &= (V, \Sigma, R, P, S) \\
V &= \{S, N_1, N_2, N_3, N_4\} \\
\Sigma &= \{a, b, c\} \\
RP &= \left\{ \begin{array}{lllllllr}
S & \rightarrow & N_1 & N_2 & N_3 & & & (0.39) \\
 & | & N_4 & & & & & (0.33) \\
 & | & N_1 & N_4 & N_3 & & & (0.22) \\
 & | & N_1 & N_1 & N_4 & N_3 & N_3 & (0.06) \\
N_1 & \rightarrow & a & & & & & (1.00) \\
N_2 & \rightarrow & c & & & & & (1.00) \\
N_3 & \rightarrow & b & & & & & (1.00) \\
N_4 & \rightarrow & N_1 & N_1 & N_2 & N_3 & N_3 & (0.61)
\end{array} \right\}
\end{aligned}
$$

Now the beam takes the form:

$$beam = \{G_{example_2}(Gain = 1.5051, unexpanded), G_{example_1}(Gain = 0, expanded)\}$$

Now, the algorithm expands $G_{example_2}$, whose candidate chunks are:

1. $N_1\ N_4\ N_3\ (-1)$
2. $N_1\ N_2\ N_3\ (-1)$
3. $N_1\ N_1\ N_4\ N_3\ (-3)$
4. $N_1\ N_1\ N_2\ N_3\ (-3)$
5. $N_1\ N_4\ N_3\ N_3\ (-3)$
6. $N_1\ N_2\ N_3\ N_3\ (-3)$
7. $N_1\ N_4\ (-2)$
8. $N_1\ N_1\ N_4\ (-3)$
9. $N_1\ N_2\ (-2)$
10. $N_1\ N_1\ N_2\ (-3)$
11. $N_4\ N_3\ (-2)$
12. $N_4\ N_3\ N_3\ (-3)$
13. $N_2\ N_3\ (-2)$
14. $N_2\ N_3\ N_3\ (-3)$
15. $N_1\ N_1\ (-2)$
16. $N_3\ N_3\ (-2)$

It can be easily seen that there is no chunk that shrinks the grammar. Let's take a look at the result of the only possible merge between S and N_4:

$$
\begin{aligned}
G_{example_3} &= (V, \Sigma, R, P, S) \\
V &= \{S, N_1, N_2, N_3\} \\
\Sigma &= \{a, b, c\} \\
RP &= \left\{ \begin{array}{lllllllr}
S & \rightarrow & N_1 & N_2 & N_3 & & & (0.39) \\
 & | & \cancel{S} & & & & & \cancel{(0.33)} \\
 & | & N_1 & S & N_3 & & & (0.22) \\
 & | & N_1 & N_1 & S & N_3 & N_3 & (0.06) \\
 & | & N_1 & N_1 & N_2 & N_3 & N_3 & (0.61) \\
N_1 & \rightarrow & a & & & & & (1.00) \\
N_2 & \rightarrow & c & & & & & (1.00) \\
N_3 & \rightarrow & b & & & & & (1.00)
\end{array} \right\}
\end{aligned}
$$

The resulting grammar is shrunk by three symbols, therefore the gain is increased, but also the likelihood is changed. The algorithm calculates the new gain:

$$\begin{aligned} Gain_3 &= Gain_2 + 3\log 2 - likelihod(S)_{old} + likelihood(S)_{new} \\ &= 1.5051 + 0.903 \\ &\quad - (0.39\log(0.39) + 0.33\log(0.33) + 0.22\log(0.22) + 0.06\log(0.06)) \\ &\quad + (0.39\log(\tfrac{0.39}{1.28}) + 0.61\log(\tfrac{0.61}{1.28}) + 0.22\log(\tfrac{0.22}{1.28}) + 0.06\log(\tfrac{0.06}{1.28})) \\ &= 2.2988 \end{aligned}$$

The resulting grammar after the merge is inserte asd $G_{example_3}$ into the beam:

$$beam = \{G_{example_3}(Gain = 2.2988, unexpanded), G_{example_2}(Gain = 1.5051, expanded)\}$$

Now, it's time to expand $G_{example_3}$, which does not have any non-terminal symbols to merge, so only chunks are possible:

1. $N_1\ N_2\ N_3\ (-1)$
2. $N_1\ N_1\ N_2\ N_3\ N_3\ (-3)$
3. $N_1\ S\ N_3\ (-1)$
4. $N_1\ N_1\ S\ N_3\ N_3\ (-3)$
5. $N_1\ N_1\ (-2)$
6. $N_3\ N_3\ (-2)$
7. $N_1\ N_2\ N_3\ N_3\ (-3)$
8. $N_1\ N_1\ N_2\ N_3\ (-3)$
9. $N_1\ S\ N_3\ N_3\ (-3)$
10. $N_1\ N_1\ S\ N_3\ (-3)$
11. $N_1\ N_2\ (-2)$
12. $N_2\ N_3\ (-2)$
13. $N_1\ S\ (-2)$
14. $S\ N_3\ (-2)$

There is no chunk that can shrink the grammar and thus the algorithm will select the least-worst among them. The "best" chunks, namely 1 and 3, are tied, but eventually they lead the search into the same grammar, so the algorithm will choose to perform the first of these chunks greedily at this step. The resulting grammar and it's gain are the following:

$$\begin{aligned} G_{example_4} &= (V, \Sigma, R, P, S) \\ V &= \{S, N_1, N_2, N_3, N_5\} \\ \Sigma &= \{a, b, c\} \\ RP &= \left\{ \begin{array}{lllllll} S & \rightarrow & N_5 & & & & (0.39) \\ & | & N_1 & S & N_3 & & (0.22) \\ & | & N_1 & N_1 & S & N_3\ \ N_3 & (0.06) \\ & | & N_1 & N_5 & N_3 & & (0.61) \\ N_1 & \rightarrow & a & & & & (1.00) \\ N_2 & \rightarrow & c & & & & (1.00) \\ N_3 & \rightarrow & b & & & & (1.00) \\ N_5 & \rightarrow & N_1 & N_2 & N_3 & & (1.00) \end{array} \right\} \end{aligned}$$

$$Gain_4 = Gain_3 + -1\log 2 = 1.9978$$

The beam now is:

$beam = \{G_{example_3}(Gain = 2.2988, expanded), G_{example_4}(Gain = 1.9978, unexpanded)\}$

Now the top grammar in the beam is the same as in the previous step $G_{example_3}$ but, since it is already expanded, the algorithm will proceed with $G_{example_4}$. Since all chunks are bad, the expansion will proceed with the merge of S and N_5:

$$
\begin{aligned}
G_{example_5} &= (V, \Sigma, R, P, S) \\
V &= \{S, N_1, N_2, N_3\} \\
\Sigma &= \{a, b, c\} \\
RP &= \left\{ \begin{array}{llllllrr}
S & \rightarrow & \cancel{S} & & & & & (\cancel{0.39}) \\
 & | & N_1 & S & N_3 & & (\cancel{0.22}) & (0.83) \\
 & | & N_1 & N_1 & S & N_3 \quad N_3 & & (0.06) \\
 & | & \cancel{N_1} & \cancel{S} & \cancel{N_3} & & & (\cancel{0.61}) \\
 & | & N_1 & N_2 & N_3 & & & (1.00) \\
N_1 & \rightarrow & a & & & & & (1.00) \\
N_2 & \rightarrow & c & & & & & (1.00) \\
N_3 & \rightarrow & b & & & & & (1.00)
\end{array} \right\}
\end{aligned}
$$

In this merge, we have an extra increase to the gain because two productions of S became identical and so they merged into one and the grammar shrank by a total of seven symbols. The new gain is:

$$
\begin{aligned}
Gain_5 &= Gain_4 + 7\log 2 - likelihod(S)_{old} + likelihood(S)_{new} \\
&= +1.9978 + 2.1070 \\
&\quad - (0.39\log(\tfrac{0.39}{1.28}) + 0.22\log(\tfrac{0.22}{1.28}) + 0.06\log(\tfrac{0.06}{1.28}) + 0.61\log(\tfrac{0.61}{1.28})) \\
&\quad + (0.83\log(\tfrac{0.83}{1.89}) + 0.06\log(\tfrac{0.06}{1.89}) + 1.0\log(\tfrac{1.0}{1.89})) \\
&= 4.0875
\end{aligned}
$$

Now, $G_{example_5}$ becomes the best grammar inside the beam:

$beam = \{G_{example_5}(Gain = 4.0875, unexpanded), G_{example_3}(Gain = 2.2988, expanded)\}$

As it can be seen, $G_{example_5}$ is close enough to the target grammar and in fact, if the algorithm chooses the "best" of the chunks which does not shrink the grammar ($N_6 \rightarrow N_1 S N_3$) and then the only possible merge, the gain will eventually increase and the resulting grammar will be the target one.

The final grammar that the algorithm will eventually return is presented below with its production probabilities normalized to one. The final gain of this grammar is

$Gain_{final} = 5.9700$.

$$
\begin{aligned}
G_{example_{final}} &= (V, \Sigma, R, P, S) \\
V &= \{S, N_1, N_2, N_3\} \\
\Sigma &= \{a, b, c\} \\
RP &= \left\{ \begin{array}{ccccccc} S & \rightarrow & N_1 & S & N_3 & (0.49) \\ & | & N_1 & N_2 & N_3 & (0.51) \\ N_1 & \rightarrow & a & & & (1.00) \\ N_2 & \rightarrow & c & & & (1.00) \\ N_3 & \rightarrow & b & & & (1.00) \end{array} \right\}
\end{aligned}
$$

In the resulting grammar, we can note the generalization that our algorithm achieved. The final grammar can produce words that fall outside the training corpus (e.g. $a\ a\ a\ a\ a\ c\ b\ b\ b\ b\ b$ with probability 0.0294).

In the original corpus, the word $a\ c\ b$ appears with a probability of seven out of eighteen; if we try to generate eighteen words using the final grammar, nine of them will probably be $a\ c\ b$ due to the 0.51 probability. This mismatch is the reason we need to recalculate the final probabilities using the Inside-Outside algorithm. After the completion of the Inside-Outside algorithm the final learned grammar will be:

$$
\begin{aligned}
G_{example_{final}} &= (V, \Sigma, R, P, S) \\
V &= \{S, N_1, N_2, N_3\} \\
\Sigma &= \{a, b, c\} \\
RP &= \left\{ \begin{array}{ccccccc} S & \rightarrow & N_1 & S & N_3 & (0.38) \\ & | & N_1 & N_2 & N_3 & (0.62) \\ N_1 & \rightarrow & a & & & (1.00) \\ N_2 & \rightarrow & c & & & (1.00) \\ N_3 & \rightarrow & b & & & (1.00) \end{array} \right\}
\end{aligned}
$$

10 Results

We will demonstrate the accuracy of our approach first by testing our learning algorithm on some textbook context-free languages (e.g. $\mathcal{L}_1 = \{a^n b^n : n \geq 1\}$) and then on some languages which belong to the class of regular languages. After that, we will present the results of our event recognition approach in two data sets. The first data set consists of artificially generated data and the second data set consists of data generated from a mission of an underwater autonomous vehicle (UAV). The complete learned PCFGs for these two data sets are listed in Appendix A.

10.1 Standard Context-Free Grammars

In this section, we use simple text-book grammars to demonstrate the capability of our learning algorithm to learn context-free grammars. Table 1 presents all the

Table 1: Learning results for several text-book PCFGs

Language	Corpus size 10	100	1000	10000
$a^n b^n, n \geq 1$	✓	✓	✓	✓
$a^n c b^n, n \geq 1$	✓	✓	✓	✓
$(ab)^n (cd)^n, n \geq 1$	✓	✓	✓	✓
Parenthesis $(()())()$	✓	✓	✓	✓
$(a + (a + (a + a))$	✓	✓	✓	✓
$wcw^R, w \in \Sigma\{a, b\}$	OG	✓	✓	✓
$ww^R, w \in \Sigma\{a, b\}$	OF	✓*	✓*	✓*
$a^{2n} b^n, n \geq 1$	✓	✓	✓	✓
$c^n (a\|d)^n, n \geq 1$	✓	✓$(\lambda = 0.4)$	✓$(\lambda = 0.3)$	✓$(\lambda = 0.4)$
$c^n (a\|d)^n a^{2m} b^m, n, m \geq 1$	OF	OG	OG	OG

Table 2: Learning results for PCFGs with different production probabilities

$a^n b^n, n \geq 1$	Corpus size 10	100	1000	10000
$S \rightarrow aSb$ (0.2) | ab (0.8)	✓	✓	✓	✓
$S \rightarrow aSb$ (0.4) | ab (0.6)	✓	✓	✓	✓
$S \rightarrow aSb$ (0.6) | ab (0.4)	BG	✓	✓	✓
$S \rightarrow aSb$ (0.8) | ab (0.2)	OG	✓	✓	✓

languages we used for testing. All these languages are CFGs, in which we assigned our own probabilities to make them PCFGs. We choose to set high probabilities to the productions that terminate the derivation process of the grammar and smaller probabilities to the ones which are capable of recursion. This choice was made because, as Table 2 indicates, when the productions that are capable of recursion have high probability, then we need a bigger learning corpus in order to learn the correct grammar.

In both tables, ✓ means that the learning algorithm returned the exact correct grammar, BG means that the learning algorithm returned a bigger, but correct, grammar compared to the original one, OG means that the resulting grammar was over-generalized and it could create words that do not belong to the original language, and OF means that the grammar was over-fitted to the training corpus and it could not create words outside the corpus that nevertheless belong to the original language.

Clearly, our learning algorithm can successfully learn text-book grammars possibly with some tuning on λ, but it failed to learn a concatenation of context-free grammars (which is also a CFG) shown in the last line of Table 1. Also, for the language $\mathcal{L} = \{ww^R, w \in \Sigma\{a, b\}\}$ marked with ✓* in the table, the initial learned grammar was wrong and after the recalculation of probabilities and the prune of a useless production, it became the correct one. Finally, it must be mentioned that for some languages and corpus sizes the regularization factor λ was shifted from the initial

Table 3: Learning results for regular languages

Language	Corpus size 10	100	1000	10000
$\mathcal{L}_{r_1}$	✓	✓	✓	✓
$\mathcal{L}_{r_2}$	OF	✓ ($\lambda = 0.04$)	✓ ($\lambda = 0.04$)	✓ ($\lambda = 0.04$)
$\mathcal{L}_{r_3}$	OG	OG	OG	OG
$\mathcal{L}_{r_4}$	✓($\lambda = 0.3$)	✓($\lambda = 0.24$)	✓($\lambda = 0.24$)	✓ ($\lambda = 0.24$)

value of 1.0 to a smaller value in order to avoid over-generalization.

Apart from context-free languages, the learning algorithm has been checked against some handwritten regular languages:

$$\mathcal{L}_{r_1} = \{a^{2n}, n \geq 1\}$$
$$\mathcal{L}_{r_2} = \{a^+b^+c^+\}$$
$$\mathcal{L}_{r_3} = \{aa^+bb^+cc^+dd^+ \mid bb^+cc^+dd^+ \mid cc^+dd^+ \mid dd^+\}$$
$$\mathcal{L}_{r_4} = \{kldc \mid dckl \mid cdzl \mid zlcd \mid dczl \mid zldc \mid cdkl \mid klcd\}$$

The results are presented in Table 3 and, as we obverse, they are not as good compared to the ones from context-free languages. These results are a strong indicator that our algorithm is biased towards over-generalization and for this reason the regularization factor λ must be chosen independently for each learning problem.

10.2 The Effect of λ Value

The merge operator affects both the prior and the likelihood and, since these two quantities are not directly comparable, the regularization factor λ is used. We conducted an experiment, where we used different values for λ in order to evaluate the effect it has on the learning process. We chose two different regular languages from the most challenging presented above, $\mathcal{L}_2$ and $\mathcal{L}_3$, in order to create two different training corpora which contained 10000 words each. Then, for each corpus, we used our learning algorithm in order to learn the corresponding grammar. Next, from each learned grammar, we derived 100000 words in order to parse them with the corresponding correct grammar. If there were words that the correct grammar was unable to parse, then the learned grammar was over-generalized. We also used the correct grammar to derive 100000 words, which were then parsed with the learned grammar. If there were words that the learned grammar failed to parse, then the learned grammar was under-generalize (over-fitted) to the training corpus. Figure 6 and Figure 7 present the results and the effect of λ on the learning process.

Ideally our learning algorithm must return a grammar which is equivalent to the target grammar and thus ideally we want both over-generalization and under-generalization (over-fitting) indicators to be zero. It turns out that for the first language (Figure 6) we can achieve the goal, when setting λ to a small value, e.g. $\lambda = 0.02$, for which both indicators become zero. However, for the second one (Figure 7), which was

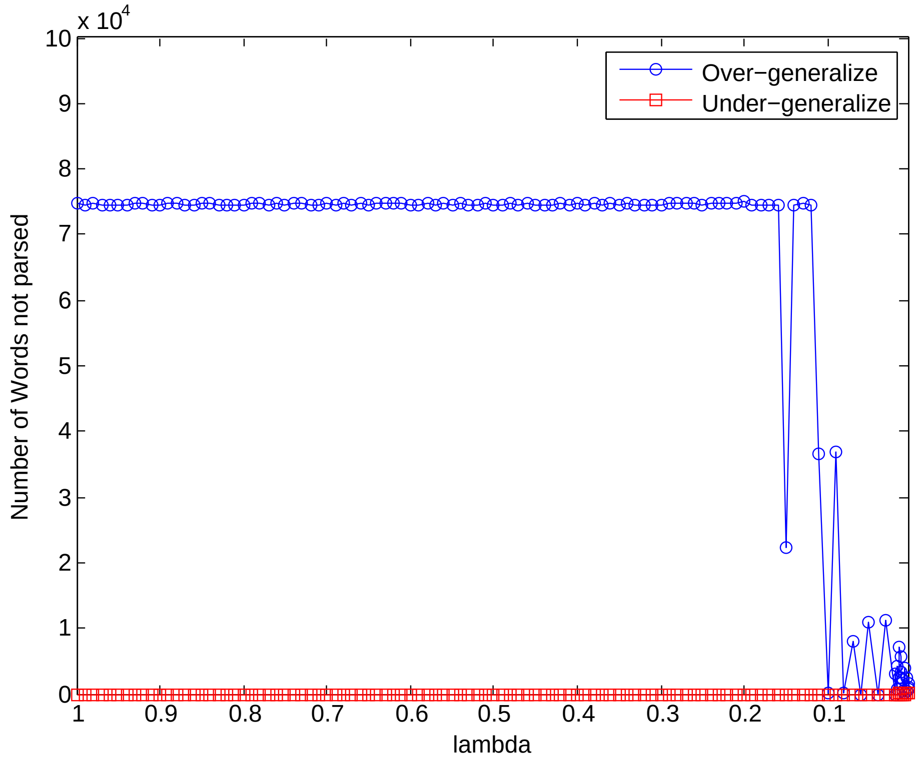


Figure 6: Results for different λ's during the learning of $\mathcal{L}_{r_2}$

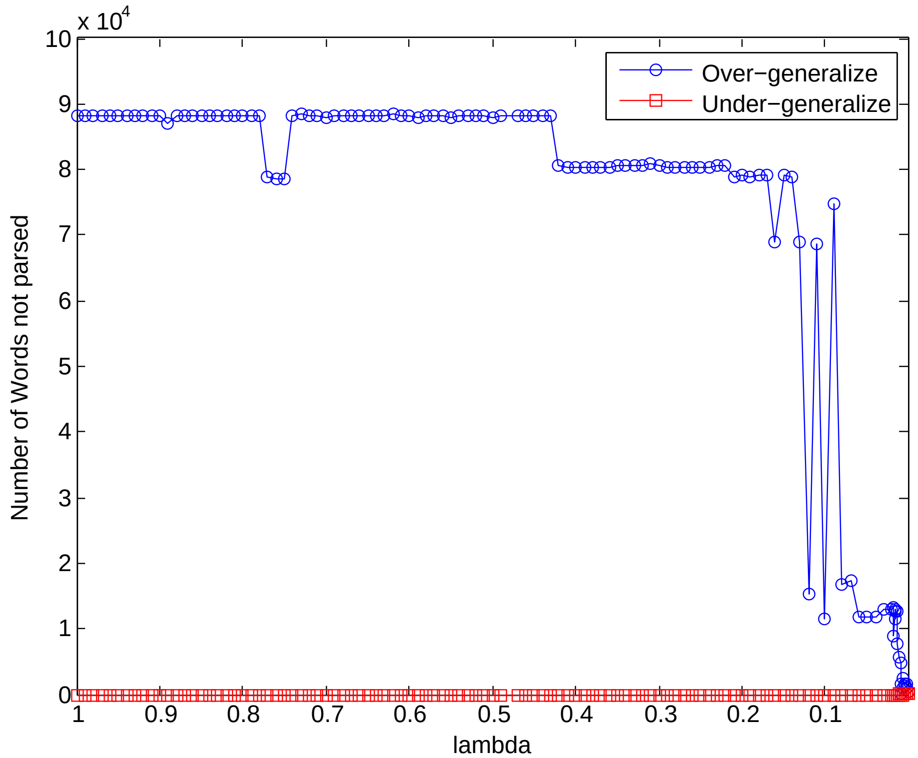


Figure 7: Results for different λ's during the learning of $\mathcal{L}_{r_3}$

more complicated, we could not find a good value for λ and our algorithm did not return a grammar equivalent to the correct one. Even though for small values of λ both indicator seem to approach zero, in reality the grammar becomes over-fitted to the training corpus.

10.3 Synthetic Data Set

In this experiment we created synthetic data for training and testing and then we added various types of noise (abnormalities) to the test set in order to construct an evaluation corpus. The function that provided the data was the cosine function and the training data are similar to the ones presented in Section 8.3 with the only change that the time window here is $[0 : 30\pi]$.

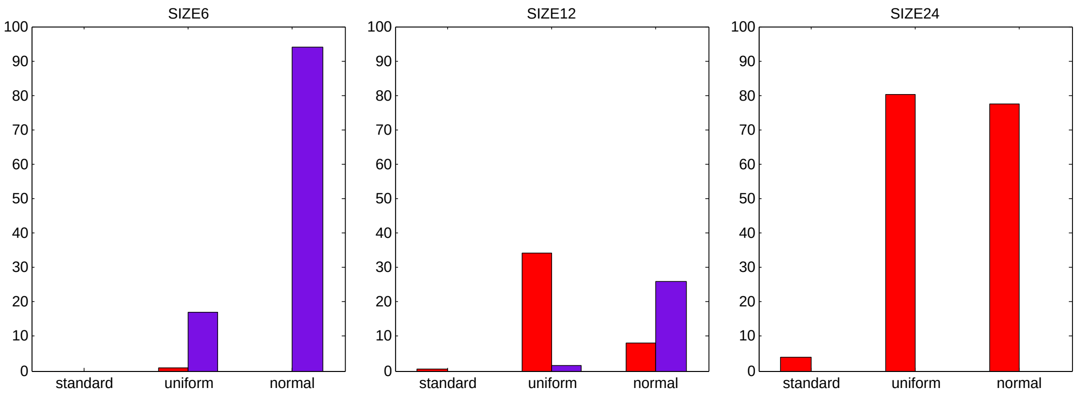
In this experiment we tested three different alphabet sizes (4, 8, 12), three different nominal word lengths (6, 12, 24), all three split approaches (standard, uniform, normal), and after experimentation we set λ to be equal to 0.05.

Figure 8 presents the results for all combinations. The accuracy of the learned PCFG is measured by calculating the number of false-positive event recognitions (magenta bar), which means that some words corresponding to abnormal events were parsed successfully and were taken as normal, and the number of false-negative event recognitions (red bar), which means that certain words corresponding to normal events failed to be parsed and were taken as abnormal. Ideally, both these numbers must be zero. The standard split method gave us the best results in all combinations of size and symbols.

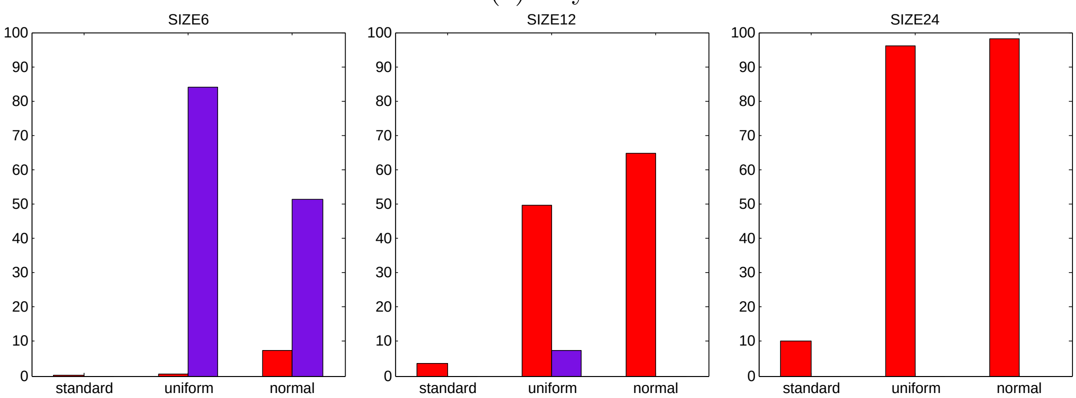
The best result was obtained when we used the standard split method, nominal word length equal to 6, and alphabet size equal to 4. For this setting, Figure 9 presents the parsing result on a distinct evaluation data set where we have identified precisely all occurrences of abnormal events (noise, shifting, and phase change). It is easy to see that our approach identifies all the abnormal events and it does not get affected neither by the shift of the cosine function nor by the change of the cosine to a sine function during the mission. On the other hand, it identifies the transition from the cosine to sine and back to cosine as abnormal events, which is correct because the training set did not include such transitions.

10.4 Noptilus AUV Data Set

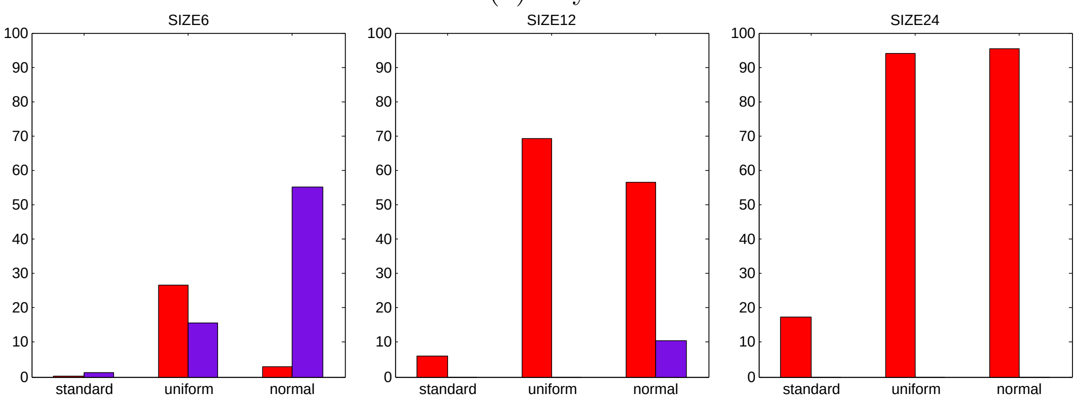
This data set came from a mission conducted with an Autonomous Underwater Vehicle (AUV) of the Noptilus project. The AUV that was used in this mission is one of those shown in Figure 10. The training data set came from a pre-specified mission and Figure 11 presents the path that the AUV followed during this normal mission. The operator of the robot repeated the normal mission, but near the end of the mission the robot was forced to maneuver in a cycle around itself in order to create an abnormal event. Figure 12 presents the path that the robot followed during the second mission, where the only difference compared to the first one is the



(a) 4 symbols



(b) 8 symbols



(c) 12 symbols

Figure 8: Results from the synthetic data set (red bars: false-negative events, magenta bars: false-positive events)

extra cycle near the end of the mission.

In order to find which sensor contains valuable data in order to recognize this event, we examined all the Inertial Measurement Unit (IMU) sensors (accelerometers, gyrometers, depth-meters). We found out that the sensor measuring the z-angle of the robot provides enough information about this event and, thus, it was the one we used to create a training and an evaluation data set. This sensor stream has high density and for this reason we down-sampled it by 100 in order to create the corresponding

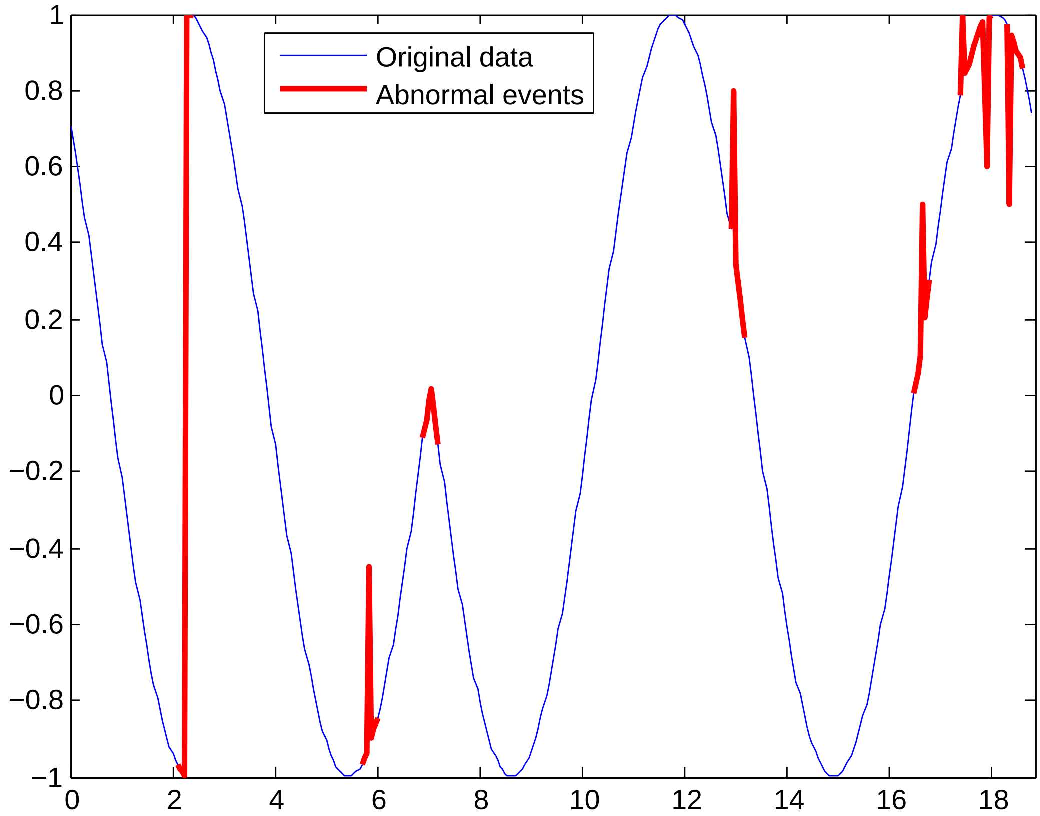


Figure 9: Evaluation data for the synthetic data set

data sets. The resulting training data set is presented in Figure 13.

For this data set we found out that the best results were obtained, when we used the following combination of parameters: standard split, nominal word size equal to 6, alphabet size equal to 12, and λ equal to 0.05. Figure 14 shows the events that our learned PCFG recognized as abnormal in the evaluation set. It can be easily seen that it correctly recognizes the beginning and the ending of the abnormal event, which have a unique footprint compared to the training set. Additionally, the PCFG recognizes some normal events as abnormal; the reason behind these false-negatives is that the training set is relatively small and does not contain many normal operation data, which can generalize the learned PCFG even more.

11 Summary

The domain of event recognition is still in its infancy, nevertheless it will gain importance in the future, as it aims at discovering discrete phenomena within vast amounts of data. Approaches based on Grammatical Inference have been little investigated, nevertheless it seems that they offer a great potential.

This research took a small step in this direction by showing that it is possible to automatically learn PCFGs appropriate for event recognition directly from data. To obtain concrete results, this work had to focus on the problem of recognizing normal

Figure 10: The Noptilus AUVs

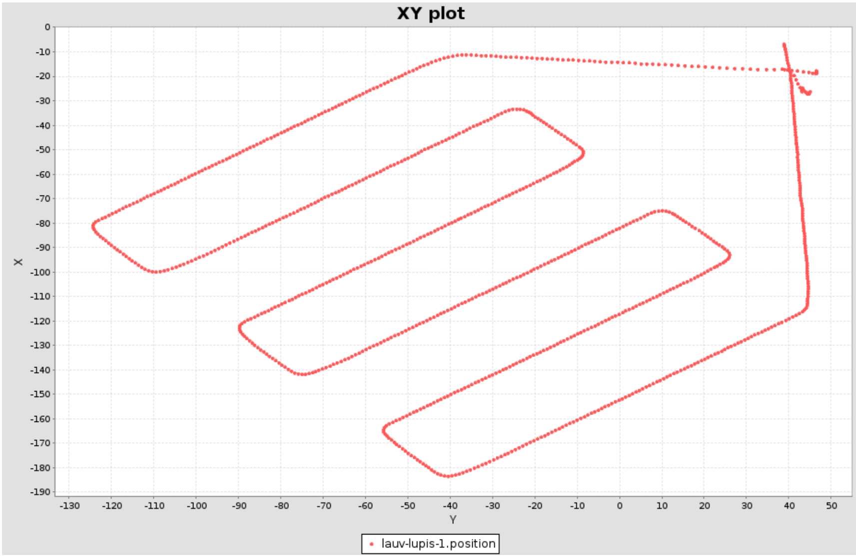


Figure 11: Path of the AUV during normal operation

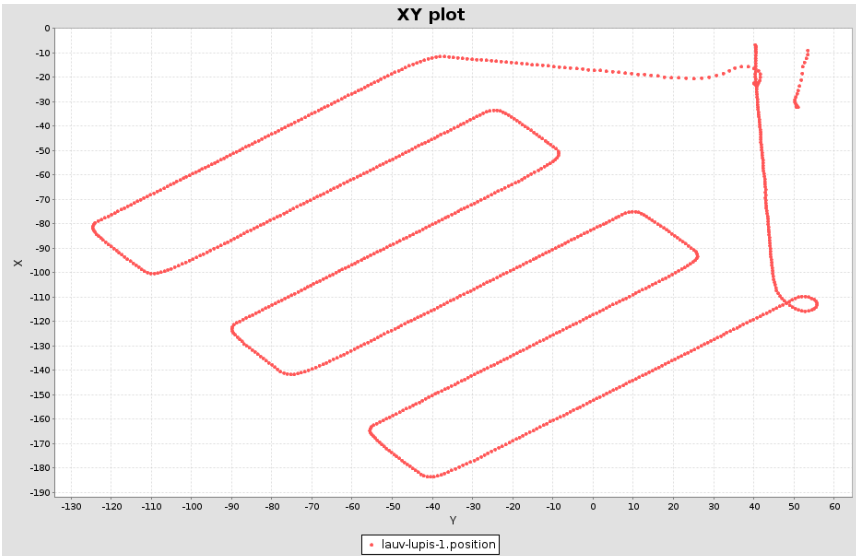


Figure 12: Path of the AUV during abnormal operation

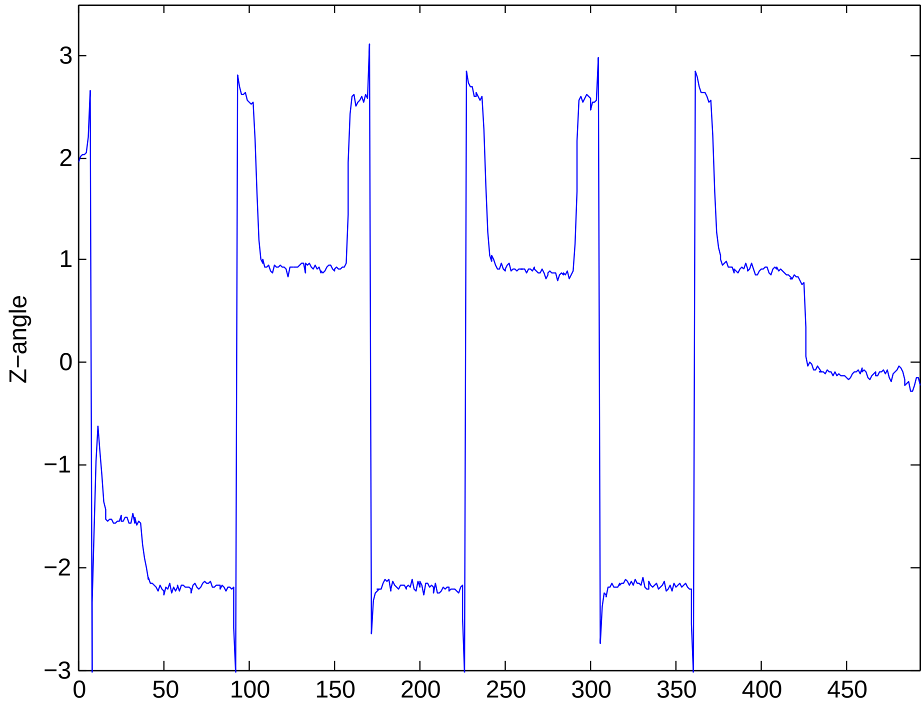


Figure 13: Data stream of the z-angle AUV sensor during normal operation

versus abnormal events due to the typical abundance of data from normal operation. The nature of the data we had to work with was totally different from typical data coming from domains, such as Natural Language Processing, where Grammatical

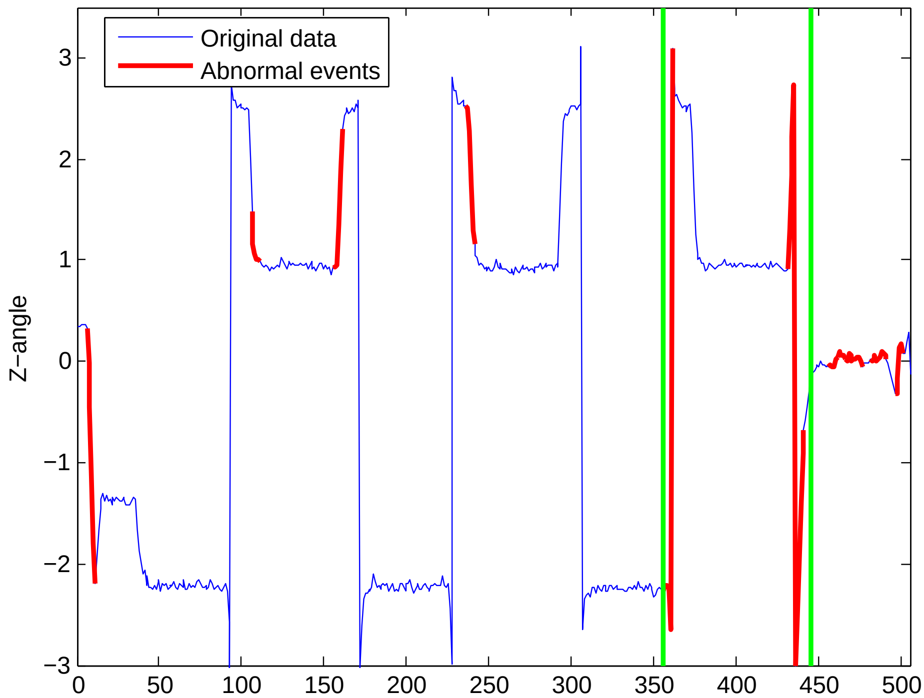


Figure 14: Data stream of the z-angle AUV sensor during abnormal operation

Inference is mostly used. The proposed procedure is quite generic and can be used to automatically construct PCFGs, which encode sensor data sequences that typically appear during normal robot operation, using recorded logs from past missions.

Leaving out the data preprocessing and the data quantization part, the process of learning a PCFG is a rather complex and challenging problem. It can be seen as an instance of structure prediction, given that a formal grammar represents an object with rich, but well-defined, structure. Due to the complexity of the problem, the proposed approach had to focus on local search methods with a small number of grammar manipulation operations. Nevertheless, in order to obtain some assurance that the proposed algorithm correctly learns all parts of a grammar structure, a significant amount of our work was dedicated on the problem of learning well-known (textbook) grammars with a variety of structural features (recursion, symmetry, repetition, etc.). The success of the proposed approach on such problems is a strong indication that the learned grammars in event recognition domains, where no ground truth is available, will likely correctly capture the underlying patterns behind the target events.

The proposed procedure was evaluated on two event recognition domains with promising results. The results indicate that our approach is capable of producing reliable PCFG-based event recognizers, which may yield some false positive and false negative signals, but in general succeed in capturing abnormalities. Despite the incorrect event reports, the proposed approach offers a filter, which can direct the attention of a mission operator to specific parts of a mission containing interesting information.

11.1 Future Work

The work in this research can be used as the base for several future research directions, some of which are listed below. Additionally, there are several ideas in our approach which can be investigated in more depth in order to produce more accurate results.

Prior probability The model we used to calculate the prior probability of a grammar may not be the best choice and in general it seems implausible that a unique optimal choice exists. Our choice often leads the search down a wrong path and, while we tried some different models, none of them led us to better results. However, there are several other ideas that can be used in order to define the prior and, thus, this is an interest area for future research.

Grammar manipulation operations In this research, we presented and used only two basic grammar manipulation operations, namely chunk and merge. In related work, we found that other operations exist, such as un-chunk and append a non-terminal greedily at the end of a production, which may have a positive effect to our local search procedure. The addition of a new operation to our approach can be easily done without major changes, as it affects only the locality of the search space.

Combined sensor streams In all our Event Recognition examples conducted with real mission data, we recognized events using only one sensor stream at a time. It would be interesting to conduct a study to find out whether more than one sensor streams can be combined to create words for a single corpus. The results of such a study could enable the recognition of more elaborate events, which require data from more than one sensor in order to be recognized.

Multi-robot event recognition It is not straightforward how the proposed approach can be applied to event recognition in a multi-robot mission. There are several challenging problems, such as the fusion of data streams from more than one robots, in order to make the recognition of a team event possible. A team event necessarily relates to data from more that one robot, for example determining if the entire team has reached a given formation. In such scenaria, it is not clear how exactly the parsing will take place. A naive idea is to have a coordinator robot which collects all data, does the parsing, and communicates back the outcome. However, it seems possible that other approaches which rely on distributed parsing or hierarchical decomposition may be more effective. Some of these issues are investigated in parallel research within Task 6.1.

References

[1] K. Lari and S. J. Young, "The estimation of stochastic context-free grammars using the inside-outside algorithm," Computer speech & language, vol. 4, no. 1, pp. 35–56, 1990.

[2] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," Journal of the Royal Statistical Society. Series B (Methodological), vol. 39, no. 1, pp. 1–38, 1977.

[3] N. Chomsky, "On certain formal properties of grammars," Information and control, vol. 2, no. 2, pp. 137–167, 1959.

[4] M. Sipser, Introduction to the Theory of Computation. Cengage Learning, 2012.

[5] H. R. Lewis and C. H. Papadimitriou, Elements of the Theory of Computation. Prentice Hall, 1997.

[6] J. Earley, "An efficient context-free parsing algorithm," Communications of the ACM, vol. 26, pp. 57–61, 1983.

[7] A. V. Aho and J. D. Ullman, The Theory of Parsing, Translation, and Compiling. Prentice Hall, 1972.

[8] C. de la Higuera, Grammatical Inference, Learning Automata and Grammars. Cambridge University Press, 2010.

[9] E. M. Gold, "Language identification in the limit," Information and control, vol. 10, no. 5, pp. 447–474, 1967.

[10] A. Stevenson and J. R. Cordy, "Grammatical inference in software engineering: An overview of the state of the art," in Software Language Engineering, ser. Lecture Notes in Computer Science, K. Czarnecki and G. Hedin, Eds. Springer, 2013, vol. 7745, pp. 204–223.

[11] B. Starkie, F. Coste, and M. van Zaanen, "The omphalos context-free language learning competition," 2004, http://www.irisa.fr/Omphalos/.

[12] A. Clark, "Learning deterministic context free grammars: the Omphalos competition," Machine Learning, vol. 66, no. 1, pp. 93–110, 2007.

[13] A. Clark, R. Eyraud, and A. Habrard, "Using contextual representations to efficiently learn context-free languages," Journal of Machine Learning Research, vol. 11, pp. 2707–2744, 2010.

[14] ——, "A note on contextual binary feature grammars," in Proceedings of the EACL 2009 Workshop on Computational Linguistic Aspects of Grammatical Inference, 2009, pp. 33–40.

[15] A. Clark and R. Eyraud, "Polynomial identification in the limit of substitutable context-free languages," Journal of Machine Learning Research, vol. 8, pp. 1725–1745, 2007.

[16] A. Clark, R. Eyraud, and A. Habrard, "A polynomial algorithm for the inference of context free languages," in Proceedings of International Colloquium on Grammatical Inference, 2008, pp. 29–42.

[17] A. Clark, C. C. Florêncio, and C. Watkins, "Languages as hyperplanes: grammatical inference with string kernels," Machine Learning, vol. 82, pp. 351–370, 2011.

[18] G. Petasis, G. Paliouras, C. D. Spyropoulos, and C. Halatsis, "eg-GRIDS: context-free grammatical inference from positive examples using genetic search," in Grammatical Inference: Algorithms and Applications, ser. Lecture Notes in Computer Science, G. Paliouras and Y. Sakakibara, Eds. Springer, 2004, vol. 3264, pp. 223–234.

[19] U. Sapkota, B. R. Bryant, and A. Sprague, "Unsupervised grammar inference using the minimum description length principle," in Machine Learning and Data Mining in Pattern Recognition, ser. Lecture Notes in Computer Science, P. Perner, Ed. Springer, 2012, vol. 7376, pp. 141–153.

[20] P. Adriaans, M. Trautwein, and M. Vervoort, "Towards high speed grammar induction on large text corpora," in SOFSEM 2000: Theory and Practice of Informatics, ser. Lecture Notes in Computer Science, V. Hlavac, K. Jeffery, and J. Wiedermann, Eds., vol. 1963, 2000, pp. 173–186.

[21] K. Tu and V. Honavar, “Unsupervised learning of probabilistic context-free grammar using iterative biclustering,” in Grammatical Inference: Algorithms and Applications, ser. Lecture Notes in Computer Science, A. Clark, F. Coste, and L. Miclet, Eds. Springer, 2008, vol. 5278, pp. 224–237.

[22] A. Stolcke, “Bayesian learning of probabilistic language models,” Ph.D. dissertation, University of California, Berkeley, 1994.

[23] S. C. Geyik and B. K. Szymanski, “Event recognition in sensor networks by means of grammatical inference,” in Proceedings of IEEE INFOCOM, 2009, pp. 900–908.

[24] A. U. Memon, “Log file categorization and anomaly analysis using grammar inference,” Master’s thesis, Queen’s University, Canada, 2008.

[25] R. G. Gallager, Principles of digital communication. Cambridge University Press, 2008.

A Learned PCFGs

A.1 PCFG for the Synthetic Data Set

```
Start Rules:
N8
All Rules:
    N4  -> e (1)

    N5  -> d (1)

    N6  -> c (1)

    N7  -> b (1)

    N8  -> N10 N19 (0.304781)
        -> N11 N13 (0.294821)
        -> N9 N13 (0.0848561)
        -> N10 N14 (0.0239594)
        -> N5 N10 N6 (0.013953)
        -> N32 N24 (0.0119522)
        -> N10 N7 N6 (0.0119522)
        -> N10 N6 N7 (0.0119522)
        -> N11 N4 N9 N9 (0.0119522)
        -> N24 N22 (0.00996015)
        -> N9 N10 (0.0139744)
        -> N16 N16 (0.0191659)
        -> N16 N14 N9 (0.00796812)
        -> N16 N16 N6 (0.0656612)
        -> N5 N22 N11 (0.00796812)
        -> N22 N4 N9 (0.0059761)
        -> N14 N10 (0.0239165)
        -> N19 N16 N14 (0.0059761)
        -> N16 N9 N11 (0.0059761)
        -> N22 N16 (0.0059761)
        -> N7 N16 N16 (0.0059761)
        -> N16 N11 N4 (0.0059761)
        -> N6 N10 N5 (0.00949365)
        -> N10 N5 N4 (0.0059761)
        -> N19 N7 N32 (0.0059761)
        -> N11 N11 N11 N5 (0.0059761)
        -> N19 N19 N14 N6 (0.0059761)
        -> N32 N19 N7 (0.0059761)
        -> N4 N10 N5 (0.0059761)
```

```
    N9  -> N5 N5 (1)

    N10 -> N32 N6 (0.144045)
        -> N19 N19 N7 (0.743619)
        -> N6 N19 N19 (0.0280611)
        -> N9 N9 N5 (0.0842746)

    N11 -> N4 N4 (1)

    N13 -> N9 N24 (0.223496)
        -> N22 N4 (0.776504)

    N14 -> N6 N6 (1)

    N16 -> N14 N6 (0.671196)
        -> N24 (0.22344)
        -> N16 N5 (0.105364)

    N19 -> N7 N7 (1)

    N22 -> N11 N11 (1)

    N24 -> N9 N5 (1)

    N32 -> N14 N14 (1)
```

A.2 PCFG for the Noptilus AUV Data Set

```
Start Rules:
N24
All Rules:
    N12 -> j (1)

    N13 -> k (1)

    N14 -> l (1)

    N15 -> n (1)

    N16 -> b (1)

    N17 -> c (1)

    N18 -> m (1)
```

```
N19 -> i (1)

N20 -> h (1)

N21 -> g (1)

N22 -> e (1)

N23 -> d (1)

N24 -> N29 N29 (0.367347)
    -> N26 N26 N26 (0.27551)
    -> N27 N27 N27 (0.122449)
    -> N25 N25 (0.0612245)
    -> N22 N22 N27 N27 (0.0102041)
    -> N32 N22 (0.0102041)
    -> N14 N25 N20 N21 (0.0102041)
    -> N15 N18 N18 N25 (0.0204082)
    -> N20 N19 N13 N25 (0.0102041)
    -> N32 N20 (0.0102041)
    -> N33 N25 N21 (0.0102041)
    -> N15 N15 N18 N25 (0.0204082)
    -> N25 N33 N18 (0.0102041)
    -> N26 N19 N12 N13 N14 (0.0102041)
    -> N20 N32 (0.0102041)
    -> N29 N18 N33 (0.0102041)
    -> N16 N17 N16 N29 (0.0102041)
    -> N13 N14 N29 N16 (0.0102041)
    -> N12 N12 N12 N12 N12 N13 (0.0102041)

N25 -> N33 N14 (0.842105)
    -> N13 N19 N20 (0.157895)

N26 -> N21 N21 (0.53795)
    -> N29 N21 (0.47205)

N27 -> N23 N23 (1)

N29 -> N15 N15 N15 (1)

N32 -> N26 N26 N21 (0.21485)
    -> N29 N26 (0.51274)
    -> N25 N33 (0.27241)

N33 -> N14 N14 (1)
```