

# e-GRIDS: Computationally Efficient Grammatical Inference from Positive Examples

Georgios Petasis †, Georgios Paliouras †, Vangelis Karkaletsis †, Constantine Halatsis § and Constantine D. Spyropoulos †

† Software and Knowledge Engineering Laboratory,  
Institute of Informatics and Telecommunications,  
National Centre for Scientific Research (N.C.S.R.) “Demokritos”,  
P.O BOX 60228, Aghia Paraskevi,  
GR-153 10, Athens, Greece.  
{petasis, paliourg, vangelis, costass}@iit.demokritos.gr

§ Department of Informatics and Telecommunications,  
University of Athens,  
TYPA Buildings, Panepistimiopolis, GR-157 84, Athens, Greece.  
halatsis@di.uoa.gr

In this paper we present a new computationally efficient algorithm for inducing context-free grammars that is able to learn from positive sample sentences. This new algorithm uses simplicity as a criterion for directing inference, and the search process of the new algorithm has been optimised by utilising the results of a theoretical analysis regarding the behaviour and complexity of the search operators. Evaluation results are presented on artificially generated data, while the scalability of the algorithm is tested on a large textual corpus. These results show that the new algorithm performs well and can infer grammars from large data sets in a reasonable amount of time.

**Keywords:** grammatical inference, context-free grammars, minimum description length, positive examples

## 1 Introduction

In this paper we present a new algorithm for inducing context-free grammars solely from positive example sentences, e-GRIDS, based on the GRIDS algorithm as it appeared in [15], which in turn is based on earlier work by Wolff [34,35] and his SNPR system. Like its predecessors, the new algorithm utilises a simplicity bias for inferring context-free grammars from positive examples only. Special attention has been given to the processing efficiency and the scalability of the algorithm to large example sets. Its inference process has been optimised using the results of a theoretical analysis regarding the dynamic behaviour of the search operators and the complexity associated with each action performed during the search process.

The vast majority of grammar inference algorithms presented in the literature share a common methodology. Based on an initial set of positive training examples, an overly specific grammar is constructed that is able to recognise only these examples. Then, a set of operators generalises this initial grammar, usually with respect to a set of *negative examples*, i.e. sentences that should not be recognised by the grammar (Figure 1). The existence of negative examples is a requirement of most algorithms, due to the need to limit the extent of generalisation, as an overly general grammar will never be refuted from a new positive example. If in a given inference step a grammar is produced whose language is larger than the unknown target language, this is irreversible since no positive example could ever supply information to detect this error. Overly general grammars can be detected if negative examples are available, since the language of such a grammar is likely to include some negative examples. Thus, a learning algorithm that uses negative examples in this manner should primarily prevent *overspecialisation (or over-fitting)*, as overgeneralisation can be controlled by the negative examples. On the other hand, a learning algorithm that has to learn solely from positive examples must prevent both *overspecialisation* and *overgeneralisation*.

The learnability of various language classes, either in the Chomsky hierarchy or not, has been extensively studied in the literature. In the two relevant reference models used in the Learning theory, namely Gold’s [12] identification in the limit model and Valiant’s [32] probably approximately correct (PAC) model, results are not encouraging, even for one of the simplest classes of formal languages, i.e. the class of regular languages. According to Gold [12], if a class of languages contains all finite languages and at least one infinite language, it is not identifiable in the limit from only positive examples. As a result, regular grammars are not learnable from positive examples in Gold’s model. Furthermore, Angluin [2] proves that context free grammars are not learnable from positive examples in polynomial time, even when the learner can pose membership queries. The situation is even worse in the PAC learning model, as it has been shown that even simple classes of languages are not PAC learnable [6,7]. However, it should be noted that these theoretical results are based on worst case scenarios. Thus, their relevance to real-world grammar induction tasks is questionable.

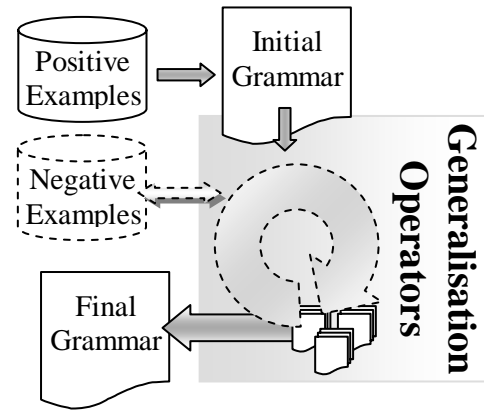
In real world problems it is not always possible to obtain a set of negative examples for training. A representative domain where negative examples are rarely available is natural language processing (*NLP*), where large sets of positive examples may be available but usually no negative examples. As the absence of negative evidence often arises in practice, two solutions have been proposed in order to alleviate this problem:

- Restricted classes of formal languages have been proven to be learnable from positive examples, such as reversible languages [3], k-testable languages [10], code regular and code linear languages [8], pure context-free languages [14,31] and strictly deterministic automata [36].
- Various heuristics aiming to avoid overgeneralisation without the use of negative examples have been proposed [25,15].

The e-GRIDS algorithm presented here belongs in the latter category and uses simplicity as a heuristic for directing the inference process, based solely on positive information. The approach of the e-GRIDS algorithm is similar to that of many grammar inference algorithms, as depicted in Figure 1. e-GRIDS is based on the GRIDS algorithm (*“GRammar Induction Driven by Simplicity”*) [15], and thus incorporates a bias towards *simple* grammars while searching the space of grammars. Viewing grammars as code, the heuristic utilised in e-GRIDS, based on Minimum Description Length (*MDL*), seeks to compress the grammar itself, as well as the encoding of the training sentences by the grammar.

However, e-GRIDS enhances GRIDS along many important dimensions. The most notable disadvantage of the GRIDS algorithm is associated with the search performed by the learning operators in the space of possible grammars. As each operator exhaustively enumerates all child grammars produced by its application on the parent grammar, GRIDS cannot be applied to training sets that contain a large number of examples or that utilise a large vocabulary. In order to eliminate this shortcoming, we have studied the dynamic behaviour of the learning operators. The results of this theoretical analysis have led to optimisations that let e-GRIDS handle large training sets. Another significant improvement relates to the introduction of a new learning operator that enhances the way e-GRIDS explores the space of possible grammars, allowing it to converge on more accurate grammars than before. Furthermore, e-GRIDS improves GRIDS’ “simplicity” criterion, providing a more accurate measurement under specific circumstances. Finally, the implementation of e-GRIDS<sup>1</sup> offers the ability to easily modify various aspects of the search process and even to introduce new search methods.

Section 2 presents the grammatical knowledge representation, the search heuristic used by e-GRIDS, and the architecture of the algorithm. Section 3 describes the learning operators, analyses their complexity and dynamic behaviour during search process, and proposes some optimisations. Sec-



**Figure 1:** Architecture of a typical grammar inference algorithm.

<sup>1</sup> The implementation of e-GRIDS (both in source and binary form) is publicly available under the GPL license at <http://www.iit.demokritos.gr/~petasis/GI.html>.

tion 4 reports an evaluation of e-GRIDS’ learning behaviour, including its scalability to large grammars. Section 5 discusses work related to our approach, whereas section 6 concludes and outlines plans for future research.

## 2 The e-GRIDS algorithm

### 2.1 Grammatical inference from positive examples

As we have already noted, grammatical inference from positive examples is more difficult than inference from example sets where negative evidence is also present. In order to compensate for the lack of negative evidence, a learning algorithm may either learn restricted classes of formal languages that are proven to be learnable from positive examples, or utilise heuristics to avoid overgeneralisation.

Like GRIDS, e-GRIDS follows the latter paradigm by incorporating a heuristic that penalises overly general grammars. This choice was partly motivated by our long-term goal of using e-GRIDS to learn natural language. For this reason, we wanted to learn grammars of a language class that can represent a large variety of linguistic phenomena. This requirement has led to the exclusion of restricted languages, as they are not expected to be able to represent naturally many of the frequently observed linguistic phenomena. The ideal candidate for natural language learning is the class of context-sensitive grammars that is sufficiently expressive to represent almost any linguistic syntactic phenomenon. However, context-sensitive grammars have been avoided, due to their various computationally undesirable properties, such as undecidability and parsing complexity. Furthermore, there is some disagreement in the literature as to whether natural languages are context-free or not<sup>2</sup>, only a few linguistic phenomena, such as cross-serial dependency, seem to need the expressiveness of context-sensitive grammars. These hard phenomena are infrequent and can be restricted in practice, allowing the language to be modelled with less expressive grammars. For these reasons, we have opted for context-free grammars instead of context-sensitive ones.

GRIDS [15] infers context-free grammars solely from positive example sets. It implements a heuristic search towards simple grammars in the space of possible grammars and uses three learning operators. The application of the operators is organised in a beam search. Being based on GRIDS, e-GRIDS shares four features with its predecessor:

- grammatical knowledge representation (context-free grammars);
- bias towards simple grammars, based on the same principle, i.e. minimum description length;
- two basic learning operators; and
- beam search approach.

However, e-GRIDS also has some notable differences with GRIDS:

- it optimises the search process by using the results of a theoretical analysis of the dynamic behaviour of the learning operators;
- it incorporates a new learning operator that can lead to more compact grammars; and
- it uses a more accurate simplicity measure based on *MDL*.

The following sections describe these points in more detail, focusing mainly on the differences of the two algorithms.

### 2.2 Knowledge Representation in e-GRIDS

Each grammar in e-GRIDS consists of a set of *context-free production (or rewrite) rules*, which are expressed using a set of *non-terminal* symbols (e.g. phrases in natural languages) and a set of *terminal symbols* (e.g. words). The set of non-terminal symbols contains a special *start symbol* (“*S*” denoting a sentence). Each production rule consists of a *head* and a *body*, in the following form:

$$\underbrace{X}_{\text{Head}} \rightarrow Y \dots \underbrace{Z}_{\text{Body}}$$

---

<sup>2</sup> Some languages, such as Swiss German and Dutch, have been shown *not* to be context free.

where  $X, Y, \dots, Z$  represent single symbols. The head (or left-hand side of the rule) consists of only one non-terminal symbol while the body (or right-hand side) may contain one or more symbols, either terminal or non-terminal ones. The interpretation of a production rule is that one can replace every occurrence of the rule head with the rule body in recognizing or generating a sentence. Like GRIDS and earlier work [33], we make the following assumptions about the form that the production rules can take:

1. No production rule can have an empty head.
2. Rules of the form " $X \rightarrow Y$ " are permitted only if the symbol " $Y$ " is a terminal symbol, i.e., a non-terminal symbol cannot be mapped to another non-terminal.
3. Every non-terminal symbol appears in the recognition or derivation of some sentence (positive example).

In addition we make the following assumption:

4. Terminal symbols are allowed only in rules of the form " $X \rightarrow Y$ ", not in rules with longer bodies. This assumption prevents terminal symbols from occurring in rules with more than one symbol in their body and is the inverse of the second assumption mentioned above.

These four assumptions do not affect in any way the representation power of a grammar, as any context-free grammar (*CFG*) can be converted to a grammar that complies with them [13]. The purpose of the fourth assumption is to allow the generation of grammars with lower description length, as will be shown in section 1. An example of a simple context-free grammar is given in Table 1.

$S \rightarrow NP \text{ VERB1}$	$ART \rightarrow the$
$S \rightarrow NP \text{ VERB2}$	$NOUN \rightarrow dog$
$NP \rightarrow ART \text{ NOUN}$	$VERB1 \rightarrow ran$
	$VERB2 \rightarrow barked$
The dog ran	
The dog barked	

**Table 1:** An example of a simple CFG and the complete set of sentences that can be generated or parsed by this grammar, as presented in [15].

More formally, a *CFG* is a quadruple  $G = (V_{NT}, V_T, P, "S")$  where  $V_{NT}$  is a set of *non-terminal symbols*,  $V_T$  is a set of *terminal symbols*,  $P$  is a set of *production rules* for generating valid sentences in the language and " $S$ "  $\in V_{NT}$  is a special symbol called the *start symbol*. The set of context-free production rules  $P$  is constructed using the start symbol, symbols from the set of *non-terminals*  $V_{NT}$ , and symbols from the set of *terminals*  $V_T$ . Each production rule is of the form  $\alpha \rightarrow \beta$ , where  $\alpha \in V_{NT}$ ,  $|\alpha| = 1$  and  $\beta \in (V_{NT} \cup V_T)^*$ .

### 2.3 e-GRIDS Search: A bias towards "simple" grammars

As e-GRIDS uses no negative evidence, an additional criterion is needed to direct the search through the space of context-free grammars and avoid overly general grammars. As we have mentioned above, this criterion provides a bias towards *simple* grammars. Viewing both the grammar and the training examples as code, a grammar A is considered simpler than a grammar B if the sum of

- (a) the number of bits required to encode grammar A, and
- (b) the number of bits required to encode the training examples as derivations of grammar A

is lower than the corresponding sum for grammar B. In this section we introduce the simplicity metric used in e-GRIDS.

As the inference process has to guard against both overly specific grammars and overgeneralisation, there are two kinds of trivial grammar we want to avoid. The first kind is a grammar that has a separate production rule for each example in the training set and does not generalise at all to new, unseen examples. The other kind of trivial grammar we want to avoid is a grammar that accepts any example. Following the GRIDS algorithm, we adopt the approach of *minimum description length (MDL)*, which directs the search process towards grammars that are compact, i.e., ones that require

few bits to be encoded, while at the same time they encode the example set in a compact way, i.e. few bits are required to encode the examples using the grammar.

The central idea behind *MDL* analysis [23] can be decomposed into four basic steps:

1. The construction of a model, based on the example set.
2. The use of this model to encode (compress) the example set and assign a length to its compressed form, usually based on notions from information theory.
3. The calculation of the length of the model, using again notions from information theory.
4. *The search for the best possible model*, corresponding to the minimisation of *the sum of the length of the compressed example set and the model length*.

In simple words, *MDL* aims at a minimally compact representation of *both the model and the data*, simultaneously. Note that *MDL* does not provide means for creating the models, i.e. the search strategy for step 4. In grammar inference, *MDL* simply offers a mechanism for comparing grammars and selecting the one that is more “compact” with respect to both the length of the grammar as well as the encoding of the training set by the grammar. Ideally we would like to know how the grammar compresses the complete language, but since this is not available, we assume that the training set is a representative sample of the language.

In order to use *MDL*, we must measure the encoding length of the grammar and of the example set, as encoded by the grammar. Assuming a context-free grammar  $G$  and a set of examples (sentences)  $T$  that can be recognised (parsed) by the grammar  $G$ , the total description length of a grammar (henceforth *model description length*, abbreviated as  $ML$ ) is the sum of two independent lengths:

- The grammar description length ( $GDL$ ), i.e. the bits required to encode the grammar rules and transmit them to a recipient who has minimal knowledge of the grammar representation, and
- the derivations description length ( $DDL$ ), i.e. the bits required to encode and transmit all examples in the set  $T$ , provided that the recipient already knows the grammar  $G$ .

$$ML = GDL + DDL$$

The first component of the  $ML$  heuristic directs the search away from the sort of trivial grammar that has a separate rule for each training sentence, as this grammar will have a large  $GDL$ . However, the same component leads to the other sort of trivial grammar, a grammar that accepts all sentences. In order to avoid this, the second component estimates the *derivation power* of the grammar (or alternatively the language of the grammar) and helps to avoid overgeneralisation by penalising general grammars. The obvious way to measure the derivation power of a grammar is to count all derivations, i.e. the sentences that can be generated. However, this is not always possible or desirable, as usually grammars generate an infinite language through recursion. Even if somebody could count all the derivations of a grammar, in most cases the result would have been a very large number compared to the length of the grammar, making the heuristic unusable. As a result, the derivation power is measured by examining the way the *training examples* are generated by the grammar. The higher the derivation power of the language, the higher its  $DDL$  is expected to be. The initial overly specific grammar is trivially best in terms of  $DDL$ , as usually there is a one-to-one correspondence between the examples and the grammar rules, i.e. its derivation power is low. On the other hand, the most general grammar has the worst score, as it involves several rules in the derivation of a single sentence, requiring substantial effort to track down all the rules involved in the generation of the sentence.

In the remaining of this subsection, we focus on the calculation of the  $GDL$  and the  $DDL$ .

### ***Grammar Description Length (GDL)***

In order to count the bits required to transmit a grammar  $G$  to a recipient, we have to decide on the way the grammar will be encoded and transmitted. Our approach is based on the separation of the rule set into three independent subsets that are transmitted sequentially. The first subset contains all the rules of the grammar  $G$  whose head is the start symbol of the grammar (start symbol subset –  $SB_1$ ). The second subset contains all the rules of the form “ $X \rightarrow Y$ ” (terminal subset –  $SB_2$ ). Finally, the third subset contains all the rules of  $G$  that are not in the first two subsets (non-terminal subset –

$SB_3$ ). The reason for this separation is the fact that the first two subsets present features that can be used to reduce the number of bits required to encode the corresponding rules. In general, the number of bits required to encode a single production rule is:

$$Bits_{Rule} = Bits_{Head} + Bits_{Body} + Bits_{End\ of\ rule}$$

In other words, the total number of bits needed for encoding a rule is the sum of the bits required to encode the rule head ( $Bits_{Head}$ ) and its body ( $Bits_{Body}$ ). Furthermore, when we transmit the grammar, a special unique symbol (e.g. “STOP”) should be appended to each rule in order to signal the end of the rule to the recipient, since rule bodies can have variable lengths. This special symbol is treated as a non-terminal requiring  $Bits_{End}$  bits to be encoded.

Supposing that the grammar  $G$  has  $A_{UNT}$  unique non-terminal symbols, excluding the special “STOP” symbol, the bits required to encode a single instance of a non-terminal is<sup>3</sup>:

$$Bits_{NT} = \log_2(A_{UNT} + 1)$$

where the additional non-terminal is the “STOP” symbol. Supposing also that the grammar has  $T$  unique terminal symbols (i.e. words), the number of bits required to encode a single instance of a terminal symbol is:

$$Bits_T = \log_2(T)$$

The rules in the start symbol subset present the special property of having as head the grammar start symbol. This property can be used to encode this subset in a more compact way, as there is no need to encode the head of each rule: this head is common for all rules and is known to the recipient. As a result, in order to encode one rule of the start symbol subset, the following expression can be used:

$$Bits_{Rule} = \left( \sum_{\substack{\forall NT \\ \text{in rule body}}} \log(A_{UNT} + 1) \right) + \log(A_{UNT} + 1)$$

Although the reduction seems to be of little significance, as we simply remove a small number of non-terminal symbols compared to the total number of symbols that need to be encoded, it has an important side effect: the total number of unique non-terminal symbols ( $A_{UNT}$ ) is reduced by one.

Regarding the terminal subset, containing the rules of the form “ $X \rightarrow Y$ ” (“ $Y$ ” being a terminal), all rules contained in the subset have a fixed length, since their body contains only one terminal symbol. Thus, no special “STOP” symbol is required to signal the end of the rule. In this case, the bits required to encode a rule from the second subset can be expressed as:

$$Bits_{Rule} = \log(A_{UNT} + 1) + \log(T)$$

The total grammar description length ( $GDL$ ) is the sum of the bits required to encode each one of the three subsets of the grammar  $G$ , plus two additional “STOP” symbols required to separate the three subsets:

---

<sup>3</sup> All logarithms in this document are assumed to be of base 2. For purposes of readability, we won’t include the logarithm base in equations.

$$\begin{array}{l}
GDL = \sum_{\substack{\forall \text{ rule in} \\ \text{Start Symbol} \\ \text{Subset}}} \left( \sum_{\substack{\forall NT \\ \text{in rule body}}} Bits_{NT} + Bits_{STOP} \right) + \left| \begin{array}{l} \text{Start Symbol Subset :} \\ \text{Body NTs + STOP} \\ \text{(Nobits needed for Head)} \end{array} \right. \\
Bits_{STOP} + \left| \text{Subset Separator} \right. \\
\sum_{\substack{\forall \text{ rule in} \\ \text{Terminal} \\ \text{Subset}}} (Bits_{NT} + Bits_T) + \left| \begin{array}{l} \text{Terminal Subset :} \\ \text{Head + Body} \\ \text{(No STOP symbol)} \\ \text{(required)} \end{array} \right. \\
Bits_{STOP} + \left| \text{Subset Separator} \right. \\
\sum_{\substack{\forall \text{ rule in} \\ \text{Non-terminal} \\ \text{Subset}}} \left( Bits_{NT} + \sum_{\substack{\forall NT \\ \text{in rule body}}} Bits_{NT} + Bits_{STOP} \right) \left| \begin{array}{l} \text{Non-Terminal Subset :} \\ \text{Head + Body NTs +} \\ \text{STOP} \end{array} \right.
\end{array} \quad (1)$$

where  $Bits_{STOP} = Bits_{NT}$

Note that the way we have chosen to encode the grammar makes four assumptions about the knowledge the recipient has of the grammar  $G$ :

- The recipient knows the set of terminal symbols.
- The recipient knows that the rule set of the grammar is divided into three subsets and the order in which these subsets are transmitted.
- The recipient knows how a rule is transmitted in each subset.
- The recipient knows the fact that two subsequent “STOP” symbols signal the beginning of a new rule subset.

An example of calculating the  $GDL$  of a simple grammar  $G$  is shown in Table 2.

$A_{UNT} = 5$ (with out “S” and “STOP”) $A_{UT} = 4$ $Bits_{NT} = \log(5+1) = 2.58$	
<b>Start Symbol Subset</b> S → NP VERB1 S → NP VERB2 STOP	$2 \cdot 2.58 + 1 \cdot 2.58 +$ $2 \cdot 2.58 + 1 \cdot 2.58 +$ $1 \cdot 2.58 +$
<b>Terminal Subset</b> ART → the NOUN → dog VERB1 → ran VERB2 → barked STOP	$1 \cdot 2.58 + 1 \cdot \log(4) +$ $1 \cdot 2.58 + 1 \cdot \log(4) +$ $1 \cdot 2.58 + 1 \cdot \log(4) +$ $1 \cdot 2.58 + 1 \cdot \log(4) +$ $1 \cdot 2.58 +$
<b>Non-Terminal Subset</b> NP → ART NOUN	$3 \cdot 2.58 + 1 \cdot 2.58$
<b>Total GDL:</b>	$16 \cdot 2.58 + 4 \cdot 2 = 49.28 \text{ Bits}$

**Table 2:** Calculating the grammar description length ( $GDL$ ) of a grammar  $G$ .

### ***Supporting terminals belonging to multiple categories***

In the original GRIDS algorithm, terminals were assumed to be classified into categories, e.g. part-of-speech categories, which were known by both the sender and the recipient. Based on this assumption the sender should only unambiguously identify the word inside each category, thus requiring  $\log(\text{Number of words in category})$  bits to encode a single terminal occurrence. However, this encoding assumes that the recipient is able to identify the category in which an incoming terminal belongs. This may be feasible when the head of the terminal rule declares the category. But terminal categories are represented by non-terminal symbols that can be modified during the search process. In such a case, the recipient has inadequate information to perform the decoding, i.e. when a terminal rule arrives having as head a new non-terminal, the recipient cannot convert it back to the original category to which the symbol initially belonged. Measuring the *ML* of a grammar in such cases is problematic, as it does not represent a correct encoding of the grammar. In order to correct this shortcoming, we must provide the missing information: in addition to the bits required to encode the terminal, the sender must also specify the category in which the terminal belongs.

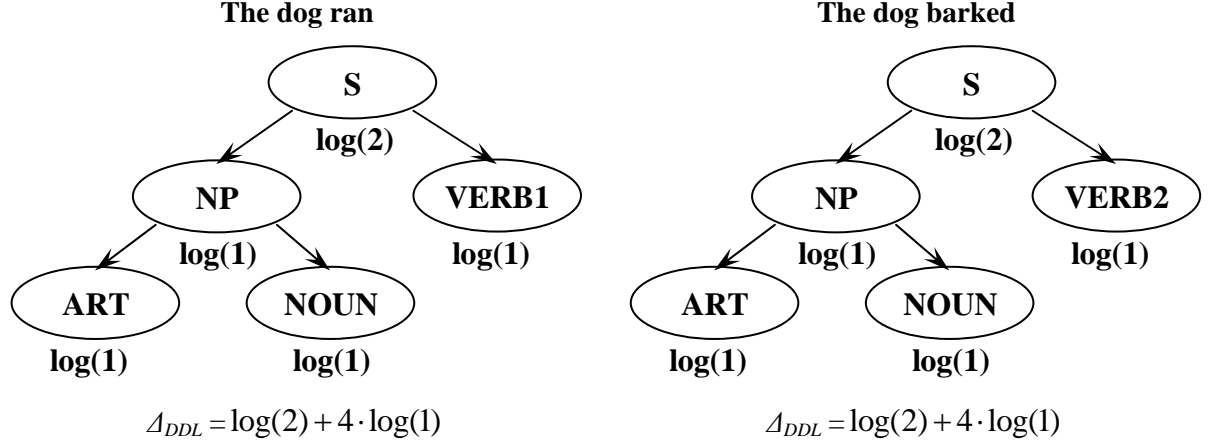
In response to this problem e-GRIDS introduces a new approach to encoding the terminal symbols of the grammar and calculating their participation in the *ML*. Unifying the way terminal symbols are treated with the way non-terminal symbols are treated, e-GRIDS assumes that all terminal symbols belong to the same set, thus requiring  $\log(\text{Number of Unique Terminals})$  to encode a single occurrence of a terminal symbol. An interesting side effect of the fact that terminals in e-GRIDS are not grouped is the ability to handle terminals that are classified into more than one category, which can be useful for example when the terminal symbols are words belonging to multiple syntactic categories.

### ***Derivations Description Length (DDL)***

The derivations description length measures the number of bits required to encode and transmit a set of sentences, as recognised (parsed) by a grammar  $G$  (or equivalently generated by that grammar), provided that the recipient already knows  $G$ . The set of sentences here is the set of training examples. Encoding the way an example is generated/parsed by a grammar  $G$  simply requires the unambiguous specification of all the rules involved in the generation/parsing of the example, i.e. specifying the complete derivation/parse tree of the example. As an example of calculating the *DDL* of a grammar, consider the complete set of the derivations of the grammar  $G$  presented in Table 1.

In order to encode the sentence “*The dog ran*”, we must specify to the recipient that the first one of the two start (“*S*”) rules must be used. In order to encode this information we need  $\log(\text{Number of Start Rules}) = \log(2)$  bits. Next, we must analyse the specified rule into its constituents, in this case “*NP*” and “*VERBI*”, and we must unambiguously specify which “*NP*” and “*VERBI*” rule to use. For encoding this information we need  $\log(1) + \log(1)$  bits, since the grammar has only one rule that starts with either “*NP*” or “*VERBI*”. Since the rule “*VERBI*” is a terminal rule, no further analysis is required. “*NP*”, on the other hand, can be further analysed into “*ART*” and “*NOUN*”. Thus, we additionally need  $\log(1) + \log(1)$  bits to resolve this ambiguity. Since neither “*NOUN*” nor “*ART*” can be further analysed, the encoding is terminated. As a result, the contribution of this sentence  $\Delta_{DDL}$  to the *DDL* is  $\log(2) + 4 \cdot \log(1)$  bits. The encoding of the second sentence requires exactly the same number of bits (see also Table 1). As a result, the *DDL* of grammar  $G$ , given this specific set of sentences, is  $2 \cdot \log(2) + 8 \cdot \log(1) = 2$  bits.

An easy and practical way to calculate the *DDL* of a grammar  $G$  is to assign to each grammar rule a *frequency*, which is simply the number of sentences from the training set in which the rule is involved in generating/parsing. The grammar  $G$  and the associated rule frequencies are shown in Table 4.



**Table 3:** Calculating the contribution to the *DDL* of two sentences.

$S \rightarrow NP \text{ VERB1}$	1
$S \rightarrow NP \text{ VERB2}$	1
$NP \rightarrow \text{ART NOUN}$	2
$\text{ART} \rightarrow \textit{the}$	2
$\text{NOUN} \rightarrow \textit{dog}$	2
$\text{VERB1} \rightarrow \textit{ran}$	1
$\text{VERB2} \rightarrow \textit{barked}$	1

**Table 4:** The grammar  $G$  and the associated rule frequencies.

In order to calculate the *DDL* of the grammar, we can simply iterate over the rules of the grammar, calculating the *DDL* of each rule. If the rule belongs to the start symbol subset, we must unambiguously distinguish the rule from all other rules in the subset. The number of bits required to encode this “disambiguation” is given by the logarithm of the number of rules that share the same head. Then, for every non-terminal symbol  $X$  in the rule body, we add to the *DDL* the number of bits required to unambiguously specify a rule that has  $X$  as head, which depends on the total number of rules that have  $X$  as head. Finally, the total number of bits required to encode a single rule should be multiplied by the rule frequency, encoding the usage of the rule in the example set.

For the other two rule subsets, the head of the rule does not need to be encoded, as the required number of bits have already been considered in the examination of rule bodies belonging to the start symbol subset. Only the non-terminal symbols in rule bodies should be considered. As a consequence, there is no need to examine rules belonging in the terminal rule subset, since their bodies contain a single terminal symbol that cannot be further analysed. In general, the *DDL* of a grammar  $G$  can be calculated as:

$$\begin{aligned}
 DDL = & \sum_{\substack{\forall \text{ rule in} \\ \text{Start Symbol Subset}}} \left( \log(H_{\text{Start Symbol}}) + \sum_{\substack{\forall X \text{ in} \\ \text{rule body}}} \log(H_X) \right) \cdot F_{\text{rule}} + \\
 & \sum_{\substack{\forall \text{ rule in} \\ \text{Non-Terminal Subset}}} \left( \sum_{\substack{\forall X \text{ in} \\ \text{rule body}}} \log(H_X) \cdot F_{\text{rule}} \right)
 \end{aligned} \tag{2}$$

where:

- $H_X = \begin{cases} \text{Number of times } X \text{ appears as Head of a rule} \\ 1 & \text{if } X \text{ does not appear as Head of a rule} \end{cases}$
- $F_{\text{rule}}$ : the rule frequency.

## 2.4 Computational Complexity of measuring the Model Length

As *MDL* guides the search process of e-GRIDS, the calculation of *ML* is a fundamental action that needs to be repeated several times. As a result, it is useful to know the complexity associated with this calculation. The process of calculating the model length of a grammar, as described by equations 2.1 and 2.2, can be realised by the simple algorithm shown in Figure 2.

```

for each Rule in Grammar {
  Count the frequency  $H_X$  of the rule head  $X$ 
}
for each Rule in Grammar {
  for each Symbol in Rule {
    Update statistics:
    Occurrence frequency of each Non Terminal
    Occurrence frequency of each Terminal
    Frequency of Non-Terminals starting rules
    ...
  }
}
Use the various statistics to measure the model length

```

**Figure 2:** Pseudo-code for calculating the *ML* of a grammar.

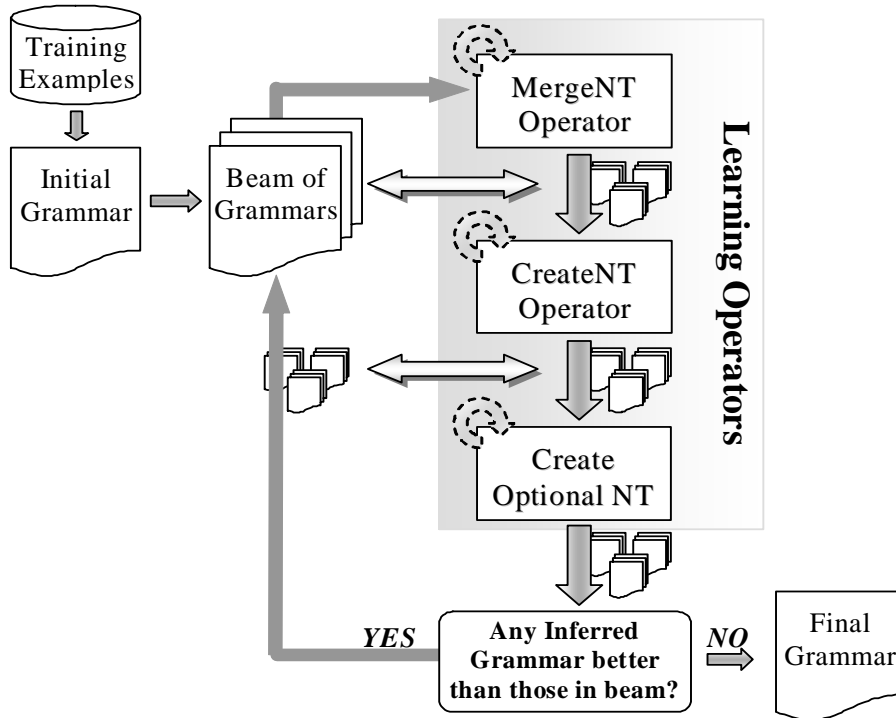
The algorithm in Figure 2 involves two iterations: the first iteration is needed exclusively for calculating the *DDL*, as it concerns the frequency of each rule head (function  $H_X$  in equation 2.2). During the second loop, all symbols within each rule are examined in order to collect the figures for evaluating equations 2.1 and 2.2 (such as the number of unique non-terminal and terminal symbols, the total occurrences of all non-terminal and terminal symbols, and how many times a non-terminal appears as the head of a rule). Finally, the model length can be calculated directly from these figures.

Thus, if  $R$  represents the number of rules in the grammar and  $S$  the average length of a rule, then the complexity of the above process is  $O(R) + O(R) \cdot O(S)$ . The number of rules in the grammar is of the same magnitude as the number of training examples (represented by  $N$ ), at least for the initial grammar, which generally has the greatest number of rules among all successor grammars, since *MDL* directs the search to more compact grammars than the initial one. The average length of a rule is a characteristic of the training examples and is very often a small number, insignificant compared to the number of training examples. As a result, the complexity of measuring the model length ( $C_{ML}$ ) of a grammar with respect to the number of training examples  $N$  is approximately linear:

$$C_{ML} = O(N) + O(N) \cdot O(S) = O(N \cdot (1 + S)) \approx O(N) \quad (3)$$

## 2.5 Architecture of e-GRIDS and the Learning Operators

As already noted, e-GRIDS induces grammars solely from positive training examples, without requiring any negative evidence. Combined with the fact that it induces context-free grammars rather than regular grammars, e-GRIDS is a good candidate for use in domains like natural language processing, where negative examples are rarely available and more expressive representations than regular grammars are required. Figure 3 summarises the architecture of e-GRIDS.



**Figure 3:** The architecture of e-GRIDS.

Like many other grammar inference algorithms, e-GRIDS uses the training sentences in order to construct an *initial, “flat” grammar*. This initial grammar is constructed by simply converting each one of the training examples into a grammatical rule, as in Table 5.

the dog ran	$\Rightarrow$	$S \rightarrow \text{THE DOG RAN}$
the dog barked		$S \rightarrow \text{THE DOG BARKED}$
		$\text{THE} \rightarrow \text{the}$
		$\text{DOG} \rightarrow \text{dog}$
		$\text{RAN} \rightarrow \text{ran}$
		$\text{BARKED} \rightarrow \text{barked}$

**Table 5:** Converting training sentences into an initial grammar.

As a result, the number of initial rules (not counting ones that replace non-terminals with terminals) is equal to the number of the training examples. This initial grammar is overly specific, as it can recognise only the sentences contained in the training example set. The parse trees created by this initial grammar are only one level deep, thus characterising the grammar as flat.

The learning process of e-GRIDS is organised as a beam search. Initially, the beam contains only the initial grammar. Having an initial hypothesis (the initial grammar) in the beam, e-GRIDS uses three learning operators in order to explore the space of context-free grammars. The “Create NT” – CreateNT operator creates a new non-terminal symbol  $X$ , which is defined as a sequence of two existing non-terminal symbols.  $X$  is defined as a new production rule that decomposes  $X$  into its two constituent symbols. The “Merge NT” – MergeNT operator merges two non-terminal symbols into a single symbol  $Y$ , thereby replacing all their appearances in the head and the body of rules by  $Y$ . Finally, the “Create Optional NT” – CreateOptionalNT operator duplicates a rule created by the CreateNT operator and appends a non-terminal symbol to the rule, thus making this symbol optional. The three operators create grammars that have either the same or greater expressiveness than their parent grammar. As the operators never remove rules from a grammar, the resulting grammars have at least the same coverage as the parent grammar, i.e. they can recognise at least the same set of sentences.

During learning, e-GRIDS iterates among three modes, by the repetitive application of the same operator at each mode. In the first mode (the “merge” mode), the algorithm considers all ways of merging non-terminal symbols by repeatedly applying the MergeNT operator. This process is repeated

for each grammar in the beam, leading to corresponding successor grammars. After all grammars in the beam have been examined, the resulting grammars are evaluated. If any of these grammars scores equally well or better than one of the grammars in the beam, the successor grammar replaces the grammar in the beam that has the lowest score. If at least one of the successor grammars manages to enter the beam, the algorithm continues in this mode.

However, if none of the successor grammars enters the beam, e-GRIDS switches to another mode. In the second mode (the “*create*” mode), e-GRIDS considers all ways of creating new terms from pairs of symbols that occur in *sequence* within the grammar, by repeatedly applying the CreateNT operator, in the same manner as described above for the “merge” mode. Again, the best successor grammars are selected and placed in the beam, thus becoming the current hypothesis for further expansion by the CreateNT operator. If none of the successor grammars enters the beam, e-GRIDS switches to the last mode, the “*create optional*” mode.

The last of the three operators, the CreateOptionalNT operator, examines all ways of duplicating a rule by the addition of an optional extra symbol at the end of the rule body. This operator is used repeatedly in exactly the same way as the previous two operators.

The algorithm continues iterating through the three modes until it is unable to produce a successor grammar that scores better than the ones in the beam. At that stage, the learning process terminates.

### 3 The Search Operators of e-GRIDS

In the following three subsections we describe in detail the search operators employed by the e-GRIDS algorithm in order to explore the space of context-free grammars. Particular attention is given to the way each operator affects the model description length of the grammar.

#### 3.1 The “Create NT” Operator

##### 3.1.1 Description of the operator

The CreateNT operator creates a new non-terminal symbol that is defined as a sequence of two existing non-terminal symbols. Renaming a sequence of two symbols “*X*” and “*Y*” into a new non-terminal symbol “*Z*”, causes the insertion into the grammar of a new rewrite rule “ $Z \rightarrow X Y$ ”, which decomposes the symbol “*Z*” into its constituents. Furthermore, all occurrences of the sequence “*X Y*” in the grammar are replaced by the symbol “*Z*”. Table 6 shows the effect of this operator.

Operator “Create NT”: Creating symbol AP1	
$NP \rightarrow ART ADJ NOUN$	$\Rightarrow NP \rightarrow ART ADJ AP1$
$NP \rightarrow ART ADJ ADJ NOUN$	$AP1 \rightarrow ADJ NOUN$

**Table 6:** The effect of the CreateNT operator, as presented in [15].

In natural language grammars, symbols created by this operator will usually represent specific phrases and clauses. The introduction of such phrases is useful when certain combinations of words (or sub-phrases) tend to occur together in sentences. The effect of this operator is a simple representation change. As the CreateNT operator simply replaces a sequence of two symbols with a new one, it does not increase or reduce the coverage of a grammar, i.e. the resulting grammar recognises exactly the same set of sentences as the initial one. Although this operator does not increase the coverage of a grammar, the representation change that it achieves is important, preparing the grammar for the application of the second operator (the “Merge NT” operator).

The effect of the application of the CreateNT operator in a grammar can be summarized as follows:

- All occurrences of the sequence “*X Y*” are substituted by the non-terminal symbol “*Z*”.
- A new rule of the form “ $Z \rightarrow X Y$ ” is appended.
- The coverage of the grammar is not affected, as the insertion of the new rule does not allow the grammar to parse new, previously unparsed, example sentences.
- The *DDL* of the grammar is not affected. No additional bits are required to unambiguously specify the new rule, since it is the only rule having as head the symbol “*Z*”.

- The *GDL* of the grammar partly increases due to the introduction of a new rule and a new non-terminal symbol. Introducing a new non-terminal symbol means that more bits are needed in order to represent each non-terminal.
- The *GDL* of the grammar is also partly reduced, since every occurrence of the sequence “*XY*” is substituted by “*Z*”.

In the following subsections we present some results regarding the complexity and the dynamic behaviour of the “CreateNT” operator.

### 3.1.2 The complexity of the “Create NT” operator mode

In this subsection we estimate the complexity of the CreateNT operator and the complexity of a complete “create” step. This step is defined as the process of applying the CreateNT operator over all possible non-terminal sequences (*bigrams*) and calculating the model length of all successor grammars. The process of performing a “create” step can be realised with the simple algorithm shown in Figure 4.

```

for each Rule in Grammar {
  for i=0, i< Rule body symbol number -1, i=i+1 {
    Store_Bigram (symbol[i], symbol[i+1])
  }
}
for each stored Bigram {
  for each Rule in Grammar {
    for i=0, i< Rule body symbol number -1, i=i+1 {
      if Bigram equals "symbol[i] symbol[i+1]" {
        => Replace "symbol[i] symbol[i+1]" with Bigram
      }
    }
  }
}
Measure Grammar Model Length [ $O(N \cdot (1+S))$ ]
}

```

**Figure 4:** Pseudo-code of a single “create” step.

As a first step, all possible sequences (bigrams) must be identified. Doing so requires an iteration over all symbols in all rule bodies. The complexity of locating all bigrams is  $O(N \cdot S)$ , assuming that the process of storing the bigrams and their occurrence frequencies is of constant complexity ( $O(1)$ ). The number of the produced bigrams (represented by  $K$ ) can be at most  $K = N \cdot S$ .

As a second step, the CreateNT operator must be applied to all bigrams. The process of applying the operator involves an iteration over all symbols in all rule bodies with a complexity of  $O(N \cdot S)$ , assuming that the process of replacing symbols in rule bodies is of constant complexity. The complexity of examining all bigrams is  $O(K \cdot N \cdot S + N \cdot (1+S))$ . As a result, the complexity of the whole step ( $C_{CreateNT}$ ) is quadratic:

$$C_{CreateNT} = O(K \cdot N \cdot S + 2 \cdot N \cdot S + N) = O(N^2 \cdot S^2 + 2 \cdot N \cdot S + N) \approx O(N^2).$$

### 3.1.3 The effect of “Create NT” on Grammar Description Length

Assuming a context-free grammar  $G$ , which has the following characteristics:

- $A_{UNT}$ : number of unique non-terminal symbols, excluding the start symbol and the special “STOP” symbol.
- $A_{UT}$ : number of unique terminal symbols.
- $A_{NT}$ : number of occurrences of all non-terminal symbols, including the start symbol of the grammar but excluding the special “STOP” symbol.

- $A_T$ : number of occurrences of all terminal symbols.
- $A_S$ : number of rules in the start symbol subset.
- $A_R$ : number of rules in the start symbol subset and the non-terminal subset.  $A_R$  implicitly measures the number of “STOP” symbols required to encode the rules of the grammar.
- $BF(X,Y)$ : number of occurrences of the bigram “X Y” that is to be substituted by “Z” in grammar  $G$ .
- $Bits_{NT} = \log(A_{UNT} + 1)$ : number of bits required to encode each occurrence of a non-terminal symbol.
- $Bits_T = \log(A_{UT})$ : number of bits required to encode each occurrence of a terminal symbol.
- $Bits_{NT}^{Fin}$ : number of bits required to encode each occurrence of a non-terminal symbol in the resulting grammar, i.e. after the operator has been applied to  $G$ .

The initial  $GDL_{In}$  of  $G$  (before the operator is applied) can be calculated as the sum of the bits required to encode all non-terminal symbols ( $A_{NT} \cdot Bits_{NT} - A_S \cdot Bits_{NT}$ ) plus the bits required to encode all terminal symbols ( $A_T \cdot Bits_T$ ) plus the bits required to encode all instances of the special “STOP” symbol ( $A_R \cdot Bits_{NT} + 2 \cdot Bits_{NT}$ ):

$$GDL_{In} = (A_{NT} + A_R - A_S + 2) \cdot Bits_{NT} + A_T \cdot Bits_T$$

The  $GDL_{Fin}$  after the operator has been applied can be calculated as:

$$GDL_{Fin} = (A_{NT} + A_R - A_S + 2 + 4 - BF(X, Y)) \cdot Bits_{NT}^{Fin} + A_T \cdot Bits_T$$

Since the application of the operator inserts a new rule of the form “ $Z \rightarrow X Y$ ”, the total number of non-terminals increases by four, the three symbols of the new rule plus the “STOP” symbol. On the other hand, every occurrence of “X Y” is substituted by “Z”, resulting in savings of  $BF(X, Y) \cdot Bits_{NT}^{Fin}$ . Since the number of unique non-terminal symbols increases by one,  $Bits_{NT}^{Fin} = \log(A_{UNT}^{Fin} + 1) = \log(A_{UNT} + 1 + 1)$ . As the  $DDL$  is not affected by this operator, the total contribution of this operator to the model length ( $\Delta ML$ ) of the grammar is equal to the change in  $GDL$  ( $\Delta GDL$ ):

$$\Delta ML = (A_{NT} + A_R - A_S + 2) \cdot \log\left(\frac{A_{UNT} + 2}{A_{UNT} + 1}\right) + (4 - BF(X, Y)) \cdot \log(A_{UNT} + 2) \quad (4)$$

### 3.1.4 Accelerating the CreateNT operator

An important property of e-GRIDS is related to computational efficiency. The candidate generation procedure of the GRIDS algorithm is an inefficient process, as it is based on the enumeration of all the grammars that can be generated by the application of an operator. Applying an operator and scoring the new grammar requires a considerable processing effort, especially in large grammars, with a few thousand rules and a few thousand terminal symbols. As a result, it is important the process of applying an operator and scoring the produced grammar to be as efficient as possible, as converging to a final grammar may require a substantial number of applications of this fundamental action.

The results of analysing the dynamic behaviour of an operator can provide valuable help in optimising its application, as these results can be used in order to forecast the model length of a generated grammar *without applying the operator to generate the grammar*. Additionally, in some cases these results may reveal information that can be used to restrict the set of symbols on which the operator must be applied, thus effectively reducing the operator invocations.

Regarding the CreateNT operator, in order to produce a successor grammar with a lower model description length than the parent one,  $\Delta ML$  has to be negative. Applying this restriction on equation 4 leads to the relation:

$$\begin{aligned} (A_{NT} + A_R - A_S + 2) \cdot \log\left(\frac{A_{UNT} + 2}{A_{UNT} + 1}\right) + (4 - BF(X, Y)) \cdot \log(A_{UNT} + 2) < 0 \Rightarrow \\ BF(X, Y) > \frac{(A_{NT} + A_R - A_S + 2) \cdot \log\left(\frac{A_{UNT} + 2}{A_{UNT} + 1}\right)}{\log(A_{UNT} + 2)} + 4 \end{aligned} \quad (5)$$

The results obtained by this analysis illustrate a property of the CreateNT operator that seems very logical and obvious: the substitution of the bigram by a new non-terminal should occur only when the two symbols of the bigram tend to appear frequently in sequence. The only parameter that is actually independent is the bigram frequency  $BF(X, Y)$ , since all the other parameters are characteristics of the parent grammar. Furthermore, the higher the bigram frequency, the greater the reduction we can achieve in the model description length of the successor grammar from the application of the operator. As a result, in order to create the  $N$  best scoring successor grammars, it suffices to apply the CreateNT operator using the  $N$  bigrams with the highest frequencies. This optimisation is very simple and drastically reduces the number of combinations that need to be examined by the CreateNT operator. Additionally, if the bigram frequency is below the threshold given by equation 5, this bigram should not be examined at all by CreateNT, as the resulting grammar will have higher model length than the parent grammar. The most important aspect of this optimisation is that it is totally equivalent to the exhaustive enumeration in terms of its effect on the overall search process. This optimisation was also used as a heuristic for guiding the search in the SNPR system [34], where it was motivated by intuition. Our theoretical analysis supports this intuition and provides the exact relation of the required bigram frequency with the structural properties of the parent grammar.

Besides the significant reduction in the bigrams that have to be examined, the theoretical analysis offers the ability to calculate the model length of the grammar directly from equation 5, without generating the child grammar. In section 3.1.2 the complexity of a mode step relevant to the CreateNT operator was calculated as  $C_{CreateNT} = O(N^2 \cdot S^2 + 2 \cdot N \cdot S + N)$ . If equation 4 is used to forecast the model length (instead of applying the operator to generate the grammar and then measure the grammar), the complexity can be reduced to  $C_{CreateNT} = O(2 \cdot N \cdot S) \approx O(N)$ .

## 3.2 The “Merge NT” Operator

### 3.2.1 Description of the operator

This operator merges two non-terminal symbols from the grammar into a new non-terminal symbol. The MergeNT operator differs from the CreateNT operator in two significant respects:

- This operator does not insert a new rule into the grammar, but modifies the heads of all corresponding rules to reflect the changes.
- There is no precondition on the relative position of the candidate symbols for merging in the grammar, i.e. the candidate symbols for merging should not necessarily form a bigram.

Merging two symbols “ $X$ ” and “ $Y$ ” into a common non-terminal symbol “ $Z$ ”, causes the substitution of all occurrences of both “ $X$ ” and “ $Y$ ” by “ $Z$ ” and the substitution of the corresponding rule heads. Table 7 shows the effect of this operator.

Operator “Merge NT”: Merging symbols AP1 and AP2	
NP → ART AP1	
NP → ART AP2	⇒
AP1 → ADJ NOUN	NP → ART AP3
AP2 → ADJ AP1	AP3 → ADJ NOUN
	AP3 → ADJ AP3

**Table 7:** The effect of the MergeNT operator, as presented in [15].

In natural language grammars, symbols created by this operator may correspond to specific word classes (e.g. nouns or verbs) and phrasal classes (e.g. noun phrases). The effect of this operator is not a simple representation change. The merging of two different symbols always increases the coverage of the grammar, as the set of sentences recognised by the grammar increases. Additionally, the use of this operator can cause rewrite rules to become identical, allowing the elimination of duplicate rules from the grammar, as shown in table 7. Another important effect of this operator is the introduction of (direct or indirect) recursion in the grammar, as can also be seen in table 7.

In order to study the effect of this operator to the model description length of a grammar, it is again useful to decompose the model description length into its two components: the grammar description length (*GDL*) and the derivations description length (*DDL*). In general, the effect of the MergeNT operator in a grammar can be summarized as follows:

- All occurrences of “*X*” and “*Y*” are substituted by the non-terminal symbol “*Z*”.
- Rules can be eliminated, since merging two symbols can lead to duplicate rules.
- The coverage of the grammar always increases.
- The *GDL* of the grammar always decreases. This reduction has two causes:
  - a) As two non-terminal symbols are merged into a single one, the number of bits required to encode non-terminals decreases.
  - b) As this operator may eliminate rules, the total number of symbols in the grammar may decrease.
- The *DDL* of the grammar can either increase or decrease due to the following causes:
  - a) The number of rewrite rules defining the new symbol “*Z*” is larger than that for either of the two merged symbols (“*X*”, “*Y*”). As a result, more bits are required to encode each occurrence of the new symbol compared to the two substituted ones.
  - b) The insertion of the new symbol “*Z*” can result in some rules becoming identical and thus being eliminated from the grammar. Identical rules are either the result of rule heads becoming identical, i.e. becoming the new symbol “*Z*”, or rule bodies becoming identical by the replacement of both “*X*” and “*Y*” by “*Z*”. An example of the latter situation is the case of the two “*NP*” rules in table 7. The elimination of duplicate rules can reduce the number of rules sharing the same head symbol (either the symbol “*Z*” or any other non-terminal symbol) and the number of bits required to encode instances of this symbol.

### 3.2.2 The complexity of the “Merge NT” operator mode

In this subsection we estimate the complexity of the MergeNT operator and the complexity of a “*merge*” step. This step is defined as the process of applying the MergeNT operator over all possible combinations of all non-terminal symbols in the grammar and calculating the model length of all resulting grammars. The process of performing a “*merge*” step can be realised with the simple algorithm in Figure 5.

```

for each Rule in Grammar {
  for each Symbol in Rule {
    Store(Symbol)
  }
}
for each Symbol_1 {
  for each Symbol_2 after Symbol_1 {
    for each Rule in Grammar {
      for each Symbol in Rule {
        if Symbol equals to either Symbol_1 or Symbol_2 {
          Replace Symbol with Symbol_1
        }
      }
    }
  }
  Measure Grammar Model Length  $O(N \cdot (1+S))$ 
}
}

```

**Figure 5:** Pseudo-code of a single step of the “merge” mode.

As a first step, all non-terminal symbols in the grammar must be identified, which requires an iteration over all symbols in all rule bodies. The complexity of this action is  $O(N \cdot S)$ , assuming that the process of storing the symbols is of constant complexity. The number of the unique non-terminal symbols can be at most  $N$  (the maximum number of rules in the grammar).

As a second step, a quadratic search is performed so as to apply the MergeNT operator over all combinations of all the non-terminal symbols, the number of which  $N^2/2$ . The process of applying the operator involves an iteration over all symbols in all rule bodies with a complexity of  $O(N \cdot S)$ , assuming that the process of replacing symbols in rule bodies is of constant complexity. The complexity of examining all symbol combinations is  $O(N^2/2 \cdot (N \cdot S + N \cdot (1+S)))$ . As a result, the complexity of the whole step ( $C_{MergeNT}$ ) is cubic:

$$C_{MergeNT} = O\left(\frac{N^3 \cdot (2 \cdot S + 1)}{2} + N \cdot S\right) \approx O(N^3)$$

### 3.2.3 The effect of “Merge NT” on Grammar Description Length

Assuming a context-free grammar  $G$ , which has the following characteristics:

- $A_{NT}$ : number of occurrences of non-terminal symbols, including the start symbol of the grammar, but excluding the special “STOP” symbol.
- $A_{UNT}$ : number of unique non-terminal symbols, excluding the start symbol and the special “STOP” symbol.
- $A_{UT}$ : number of unique terminal symbols.
- $A_T$ : number of occurrences of terminal symbols.
- $A_S$ : number of rules in the start symbol subset.
- $A_R$ : number of rules in the start symbol subset and the non-terminal subset.  $A_R$  implicitly measures the number of “STOP” symbols required to encode the rules of the grammar.
- $Bits_{NT} = \log(A_{UNT} + 1)$ : number of bits required to encode each occurrence of a non-terminal symbol.
- $Bits_T = \log(A_{UT})$ : number of bits required to encode each occurrence of a terminal symbol.
- $Bits_{NT}^{Fin}$ : number of bits required to encode each occurrence of a non-terminal symbol in the resulting grammar, i.e. after the operator has been applied to  $G$ .

As was the case with the CreateNT operator, the initial  $GDL$  of  $G$  (before the operator is applied) can be calculated as follows:

$$GDL_{In} = (A_{NT} + A_R - A_S + 2) \cdot \log(A_{UNT} + 1) + A_T \cdot \log(A_{UT})$$

Similarly, the  $GDL$  after the operator has been applied can be calculated as:

$$GDL_{Fin} = (A_{NT} + A_R - A_S + 2) \cdot \log(A_{UNT}) + A_T \cdot \log(A_{UT}) - E$$

The first term of the above equation represents the bits required to encode all non-terminals. Since the application of the operator has substituted every occurrence of either “X” or “Y” with “Z”, the total number of unique non-terminal symbols has decreased by one. As a result,  $Bits_{NT}^{Fin} = \log(A_{UNT})$ . The term  $E$  represents the expected reduction in  $GDL$  that can occur if the application of this operator leads to duplicate rules, which should be removed. For each rule that is eliminated from the grammar  $G$ , we have to remove the bits required to encode its head, its body and the “STOP” symbol:

$$E = \sum_{j \in \Omega_1} (L_j + 1) \cdot \log(A_{UNT}) + |\Omega_2| \cdot (\log(A_{UNT}) + \log(A_{UT})) + \sum_{j \in \Omega_3} (L_j + 2) \cdot \log(A_{UNT}) \quad (6a)$$

where:

- $\Omega_1$ : the set of the rules from the start symbol subset that are eliminated from  $G$ .
- $\Omega_2$ : the set of the rules from the terminal subset that are eliminated from  $G$ .  $|\Omega_2|$  is the number of rules in  $\Omega_2$ .
- $\Omega_3$ : the set of the rules from the non-terminal subset that are eliminated from  $G$ .
- $L_j$ : the number of non-terminal symbols in the body of rule  $j$ .

In equation 3.3, the term  $(L_j + 1) \cdot \log(A_{UNT})$  represents the bits required to encode the non-terminal symbols in the rule body (plus the “STOP” symbol) of a rule from the start symbol subset. The term  $\log(A_{UNT}) + \log(A_{UT})$  represents the bits required to encode the head and the terminal symbol in the body of a rule from the terminal subset. The last term  $(L_j + 2) \cdot \log(A_{UNT})$  represents the bits required to encode the head and the non-terminal symbols of the body (plus the “STOP” symbol) of a rule from the non-terminal subset.

The change  $\Delta_{GDL}$  of the  $GDL$  due to this operator is:

$$\Delta_{GDL} = (A_{NT} + A_R - A_S + 2) \cdot \log\left(\frac{A_{UNT}}{A_{UNT} + 1}\right) - \left( \sum_{j \in \Omega_1} (L_j + 1) \cdot \log(A_{UNT}) + |\Omega_2| \cdot (\log(A_{UNT}) + \log(A_{UT})) + \sum_{j \in \Omega_3} (L_j + 2) \cdot \log(A_{UNT}) \right) \quad (6b)$$

The first term in equation 3.4 is negative and can be considered “constant”, as it depends on the characteristics of the initial grammar, independently of the specific two symbols that are merged. The second term in equation 3.4 is also always negative (due to the minus sign). As a result, the reduction in  $GDL$  is larger as the number of the duplicate rules eliminated from the grammar and their length increase.

### 3.2.4 The effect of “Merge NT” on Derivations Description Length

Given a grammar  $G$  and a set of sentences recognised by  $G$ , the calculation of the  $DDL$  can be calculated as follows:

$$DDL = \sum_{\substack{\forall \text{ rule in} \\ \text{Start Symbol Subset}}} \left( \log(H_{\text{Start Symbol}}) + \sum_{\substack{\forall X \text{ in} \\ \text{rule body}}} \log(H_X) \right) \cdot F_j^S + \sum_{\substack{\forall \text{ rule in} \\ \text{Non-Terminal Subset}}} \left( \sum_{\substack{\forall X \text{ in} \\ \text{rule body}}} \log(H_X) \cdot F_j^S \right) \quad (6c)$$

where:

- $X$  represents each non-terminal symbol in the head and body of a rule  $j$ .
- $H_X = \begin{cases} \text{Number of times } X \text{ appears as Head of a rule} \\ 1 & \text{if } X \text{ does not appear as Head of a rule} \end{cases}$
- $F_j^S$  represents the number of sentences involving rule  $j$  in their generation/parsing.

As we are mainly interested in studying the effect that this operator has on grammar  $G$ , we calculate directly the change  $\Delta_{DDL}$  in  $DDL$  due to this operator. In order to calculate this change, we divide the effect of this operator into two “phenomena”, which will be studied independently of each other. The first “phenomenon” is the substitution of all “ $X$ ” or “ $Y$ ” rule heads by “ $Z$ ”, without eliminating any duplicate rules that may appear. We will refer to this event as “merge of sets of rules”. The second “phenomenon” is the elimination of duplicate rules. As explained, the application of the MergeNT operator can lead to duplicate rules in two ways. Rules sharing the same bodies but their heads being either “ $X$ ” or “ $Y$ ” will become identical when their heads are substituted by the new symbol “ $Z$ ”. Typical examples are terminal rules, where one terminal may belong to two categories. One of the two rules must be eliminated if the non-terminal symbols representing the two categories are merged. The second situation where duplicate rules occur is the substitution of “ $X$ ”, “ $Y$ ” by “ $Z$ ” in rule bodies. It is possible for these two mechanisms to occur simultaneously on a single rule. In the following paragraphs we examine the effect of each of the two phenomena on  $\Delta_{DDL}$ .

#### Merging sets of rules

As discussed above, the MergeNT operator does not insert a new rule into the grammar. Instead, the operator modifies the heads of some rules to reflect the required changes. This is shown in Table 8.

<b>Operator “Merge NT”: Merging symbols X and Y</b>	
<i>Initial Grammar (<math>G_{In}</math>)</i>	<i>Final Grammar (<math>G_{Fin}</math>)</i>
$X \rightarrow A_1 B_1 C_1$	$Z \rightarrow A_1 B_1 C_1$
$X \rightarrow A_2 B_2 C_2$	$Z \rightarrow A_2 B_2 C_2$
$X \rightarrow A_3 B_3 C_3$	$Z \rightarrow A_3 B_3 C_3$
$Y \rightarrow D_1 E_1 F_1$	$Z \rightarrow D_1 E_1 F_1$
$Y \rightarrow D_2 E_2 F_2$	$Z \rightarrow D_2 E_2 F_2$
$K \rightarrow L_1 M_1 N_1$	$K \rightarrow L_1 M_1 N_1$
...	...

**Table 8:** Merging sets of rewrite rules.

In the initial grammar  $G_{In}$  of table 8,  $\log(3)$  bits are required in order to unambiguously identify each rule having as head the symbol “ $X$ ”, while  $\log(2)$  bits are required for identifying a rule having as head the symbol “ $Y$ ”. In the resulting grammar  $G_{Fin}$ , identifying a rule that has symbol “ $Z$ ” as head requires  $\log(5)$  bits. In the general case, the bits required to identify each rule having the new symbol “ $Z$ ” as head is:

$$Bits_Z = \log(F_X + F_Y)$$

where:

- $F_X$  represents the number of rules that have symbol “X” as head and
- $F_Y$  represents the number of rules that have symbol “Y” as head.

In order to calculate the contribution of the symbols X and Y to the *DDL* of the initial grammar  $G_{In}$  we need to count the number of times each symbol appears in rule bodies. As a result, the total contribution of both “X” and “Y” to the *DDL* is:

$$C_{In} = \alpha \cdot \log(F_X) + \beta \cdot \log(F_Y)$$

where:

- $\alpha = \sum_{j \in G_{In}} N_j^X \cdot F_j^S$ ,  $\beta = \sum_{j \in G_{In}} N_j^Y \cdot F_j^S$
- $F_j^S$  represents the number of sentences involving rule  $j$  in their generation/parsing.
- $j$  represents a single rule of the grammar  $G_{In}$ .
- $N_j^X$  represents the number of occurrences of the symbol X in the body of rule  $j$ .
- $N_j^Y$  represents the number of occurrences of the symbol Y in the body of rule  $j$ .

On the other hand, the contribution of the new symbol Z to the *DDL* of  $G_{Fin}$  can be calculated as:

$$C_{Fin} = (\alpha + \beta) \cdot \log(F_X + F_Y)$$

Thus, the change  $\Delta C$  of the *DDL* due to the merging of the two rule sets is:

$$\Delta C = \alpha \cdot \log\left(1 + \frac{F_Y}{F_X}\right) + \beta \cdot \log\left(1 + \frac{F_X}{F_Y}\right). \quad (6d)$$

### Removing duplicate rules

The elimination of rules that have become identical is a side effect of applying the MergeNT operator and largely depends on the characteristics of the grammar under examination. In cases where such elimination does occur, the total contribution to the *DDL* is:

$$\Delta C = \sum_{j \in G_{Fin}} \left( \sum_{k \in \Theta, j} M_k \right) \cdot F_j^S \quad (6e)$$

where:

- $\Theta$  is the set of head symbols of rules that were removed.
- $k$  represents each symbol in the body of a rule  $j$  that is also a member of the set  $\Theta$ . Note that since  $k$  is a member of  $\Theta$  it cannot be the new symbol Z.
- $M_k$  represents the reduction of the number of bits required to encode a single occurrence of the symbol  $k$ .  $\left( M_k = \log\left(\frac{\text{Number of rules having } k \text{ as head in } G_{Fin}}{\text{Number of rules having } k \text{ as head in } G_{In}}\right) \right)$ . Note that  $M_k$  is always negative.
- $F_j^S$  represents the number of sentences involving rule  $j$  in their generation.

### 3.2.5 Total contribution of “Merge NT” to the Model Description Length

The total contribution of the *MergeNT* operator to the model description length of a grammar can be calculated by combining equations 6b, 6d and 6e:

$$\begin{aligned}
 \Delta ML = & (A_{NT} + A_R - A_S + 2) \cdot \log\left(\frac{A_{UNT}}{A_{UNT} + 1}\right) - \\
 & \left( \sum_{j \in \Omega_1} (L_j + 1) \cdot \log(A_{UNT}) + |\Omega_2| \cdot (\log(A_{UNT}) + \log(A_{UT})) + \right. \\
 & \left. \sum_{j \in \Omega_3} (L_j + 2) \cdot \log(A_{UNT}) \right) + \\
 & \left( \sum_{j \in G_m} N_j^X \cdot F_j^S \right) \cdot \log\left(1 + \frac{F_Y}{F_X}\right) + \left( \sum_{j \in G_m} N_j^Y \cdot F_j^S \right) \cdot \log\left(1 + \frac{F_X}{F_Y}\right) + \\
 & \sum_{j \in G_{fin}} \left( \sum_{k \in \Theta, j} M_k \right) \cdot F_j^S
 \end{aligned} \tag{6}$$

The first two terms of equation 6 correspond to the change in *GDL*. As explained in section 3.2.3, the reduction in *GDL* is larger as the number of the duplicate rules eliminated from the grammar and their length increase. The third and fourth terms of equation 6 correspond to the change in *DDL*. The third term represents the merging of sets of rules and is always positive. The independent parameters are four, the relative frequency of the two symbols “X” and “Y” as head of rules  $\left(\alpha = \frac{F_X}{F_Y}\right)$ , the number of times “X” and “Y” appear in rule bodies  $(N_j^X, N_j^Y)$  and the rule frequencies  $F_j^S$ . The two logarithms of the third term in equation 6 reach a minimum when  $\alpha=1$  ( $F_X=F_Y$ ). Thus, this term is small when  $\alpha$  equals one and “X” and “Y” do not appear frequently in rule bodies that are associated with high frequencies. Finally, the fourth term represents the elimination of duplicate rules and is always negative or zero, depending on the existence of duplicate rules or not.

### 3.2.6 Accelerating the MergeNT operator

The MergeNT operator can produce better scoring successor grammars than the parent only if  $\Delta ML < 0$ . The first term of equation 6 is constant (for a given parent grammar  $G$ ) and always negative. The second and fourth terms are also always negative or zero, depending on whether duplicate rules exist in the new grammar. Finally, the third term is always positive. Thus, in order to satisfy  $\Delta ML < 0$ , the sum of the first, the second, and the fourth terms must be greater, in absolute terms, than the third term.

Intuition in this case suggests the application of the MergeNT to non-terminals that frequently appear in similar contexts. In fact such a heuristic was used in SNPR [34] for reducing the number of grammars that have to be searched. “Similar context” can be thought of as rules with small differences other than the two symbols that will be merged, highly probable to become duplicate and thus eliminated from the grammar when the two symbols are actually merged. As a result, this “common sense” property of the MergeNT operator can be associated with duplicate rule elimination in our framework. However, from equation 6 we can easily conclude that although rule elimination is an important factor, it is not the only one. An important condition for the successful application of this operator is a trade-off between the number, frequency and length of the eliminated duplicate rules and the *distribution of the two merged symbols in the grammar* (third term of equation 6). Thus, the existence in similar contexts cannot be used as a criterion for directing the search, as it is based primarily on rule elimination, but fails to capture an equally significant factor: the relative frequencies of the two symbols as rule heads and their distribution in the grammar. Instead of this “incorrect” heuristic, e-GRIDS uses equation 6 to direct the search by calculating the model description length without actually generating

the corresponding grammars. The calculation of equation 6 is considerably faster than producing and scoring the corresponding grammar.

In section 3.2.2 the complexity of an MergeNT mode step was found to be  $C_{MergeNT} = O(1/2 \cdot N^3 \cdot (2 \cdot S + 1) + N \cdot S)$ . Forecasting the model length instead of generating and measuring the child grammar presents a complexity of  $C_{MergeNT} = O(1/2 \cdot N^2 \cdot M + N \cdot S)$ , where M is a count of the rules that will be eliminated from the grammar ( $M = |\Omega_1| + |\Omega_2| + |\Omega_3|$ ) which is usually a small number. However, a mode that forecasts the model length is still a quadratic process, which may be a problem if the number of training instances is large. In such cases additional heuristics must be applied that possible combine the idea of “similar context” with the overall presence of the various symbols in the grammar.

### 3.3 “Create Optional NT” Operator

e-GRIDS introduces a new operator, the “Create Optional NT”, and the relevant mode in the search process. The introduction of this new operator was necessary as the existing operators were unable to infer correctly some types of grammars, often converging to slightly less general grammars than the correct one. This inefficiency has been mainly attributed to the fact that the older two operators could not create rules that contain an optional symbol, i.e. a symbol that either exists or not but only a single occurrence can be expanded by the rule if the symbol exists. Such “optional” symbols could exist in learned grammars only as oddments in rewrite rules starting with the start symbol or through recursion. Leaving these symbols in start symbol rules prevents them from merging (and thus leading to more compact grammars) and the introduction of recursion to describe these symbols leads to more general grammars which often get rejected. The introduction of the CreateOptionalNT operator tries to alleviate this inefficiency by inserting rules in the grammar that constitute symbols optional, rules that couldn’t have been possible to have been inserted by the rest of the operators.

#### 3.3.1 Description of the operator

e-GRIDS introduces a new operator, the “Create Optional NT”, and the relevant mode in the search process. The introduction of this new operator was necessary as the existing operators were unable to infer correctly some types of grammars, often converging to less general grammars than the correct one. This inefficiency has been mainly attributed to the fact that the older two operators could not create rules that contain an optional symbol, i.e. small variations of existing rules.

Create Optional NT – CreateOptionalNT seeks to expand a rule created by the CreateNT operator by attaching an additional non-terminal symbol from the existing ones. This expansion does not affect the original rule, as the CreateOptionalNT operator appends a new rule that is a duplicate of the original rule, with a non-terminal symbol appended at the end of the rule body, as shown in Table 9.

<b>Operator “Create Optional NT”: Making optional symbol ADJ</b>	
NP → AP1 NOUN	NP → AP1 NOUN
NP → AP1 ADJ NOUN	NP → AP1 ADJ NOUN
NP → AP1 ADJ ADJ NOUN	AP1 → ART ADJ
AP1 → ART ADJ	<b>AP1 → ART ADJ ADJ</b>
X → A1 AP1	X → A1 AP1
Y → X ADJ NOUN	Y → X NOUN

**Table 9:** The effect of the CreateOptionalNT operator.

The effect of this operator in the grammar is that the appended symbol becomes optional with respect to the original rule, as created by the CreateNT operator. The fact that the CreateOptionalNT operator does not alter the original rule and appends a new rule with the same rule head is a generalisation step.

In theory, the effect of the CreateOptionalNT operator could be achieved by the CreateNT and MergeNT operators, as shown in Table 10.

<b>Operator “Create NT”: Creating symbol AP2</b>		
$AP1 \rightarrow ART\ ADJ$	$\Rightarrow$	$AP1 \rightarrow ART\ ADJ$ $AP2 \rightarrow ART\ ADJ\ ADJ$
<b>Operator “Merge NT”: Merging symbols AP1 &amp; AP2</b>		
$AP1 \rightarrow ART\ ADJ$	$\Rightarrow$	$AP1 \rightarrow ART\ ADJ$
$AP2 \rightarrow ART\ ADJ\ ADJ$	$\Rightarrow$	$AP1 \rightarrow ART\ ADJ\ ADJ$

**Table 10:** Analysing the effect of the CreateOptionalNT operator.

However, in practice this is not possible for the following reasons:

- CreateNT cannot create a rule with three symbols in the rule body, as it operates solely on bigrams.
- The creation of “AP1” by CreateNT, the only operator that creates new rules, will eliminate all occurrences of the bigram “ART ADJ” from the grammar. As a result, a rule of the form “AP2  $\rightarrow$  ART ADJ ADJ” cannot exist, so that MergeNT can lead to the grammar of Table 11.

The effect of this operator is not a simple representation change, as the addition of a new rule that shares the same rule head with an existing rule always increases the coverage of the grammar. Additionally, the use of this operator can cause rewrite rules to become identical, allowing the elimination of duplicate rules from the grammar.

$S \rightarrow \dots X\ Y \dots$	$\Rightarrow$	$S \rightarrow \dots X \dots$
$S \rightarrow \dots Z\ Y \dots$		$S \rightarrow \dots Z \dots$
$S \rightarrow \dots W\ Y \dots$		$S \rightarrow \dots W \dots$
$W \rightarrow \dots X$		$W \rightarrow \dots X$
$Z \rightarrow \dots W$		$Z \rightarrow \dots W$
$X \rightarrow X1\ X2$		$X \rightarrow X1\ X2$
		$X \rightarrow X1\ X2\ Y$

**Table 11:** Replacing all occurrences of  $W_X Y$ , where  $W_X$  can be expanded to end with the symbol  $X$ .

In general, the effect of the CreateOptionalNT operator in a grammar can be summarized as follows (assuming that the rule “ $X \rightarrow X1\ X2$ ” is to be augmented with the non-terminal symbol “ $Y$ ”):

- All occurrences of the sequence “ $X\ Y$ ” are substituted by the non-terminal symbol “ $X$ ”.
- All occurrences of the sequence  $W_X Y$  are substituted by  $X$ , if the body of the rule of which  $W_X$  is the head ends with  $X$ , or the rule’s last body symbol can be expanded to end with  $X$ . (An example can be seen in Table 11.)
- A new rule of the form “ $X \rightarrow X1\ X2\ Y$ ” is added to the grammar.
- Rules may be eliminated from the grammar, since bigram substitutions can lead to duplicate rules.
- The coverage of the grammar always increases.
- The *GDL* of the grammar is modified, as:
  - a) A new rule of the form “ $X \rightarrow X1\ X2\ Y$ ” is appended.
  - b) The elimination of the symbol “ $Y$ ” from some rule bodies can cause rules to be merged, thus reducing the total number of symbol occurrences in the grammar.
- The *DDL* of the grammar is also modified due to the following causes:
  - a) The number of rewrite rules defining the symbol “ $X$ ” is increased by one. As a result, more bits are required to encode each occurrence of the symbol “ $X$ ” than before, thus increasing the *DDL*.
  - b) Rule elimination can potentially decrease the number of times that some non-terminals (including “ $X$ ”) appear as rule heads, reducing the number of bits required to encode each occurrence of these symbols and thus reducing the *DDL*.

### 3.3.2 The complexity of the “Create Optional NT” operator mode

In this paragraph we estimate the complexity of the CreateOptionalNT operator and the complexity of a “create optional” step. This step is defined as the process of applying the CreateOptionalNT operator over all possible bigrams whose first symbol is a non-terminal symbol created by the CreateNT operator and calculating the model length of all successor grammars. The process of performing a single “create optional” step can be realised with the algorithm in Figure 6.

```

for each Rule in Grammar {
  for i=0, i< Rule body symbol number, i=i+1 {
    if symbol[i] created by the CreateNT operator {
      Store_Bigram (symbol[i], symbol[i+1])
    }
  }
}
for each stored Bigram {
  SH = Bigram(0)
  Y = Bigram(1)
  do {
    for each Rule in Grammar {
      if last symbol in rule body is in SH {
        Append rule head to SH
      }
    }
  } while SH modified
  for each symbol X in SH {
    for each Rule in Grammar {
      for each Symbol in Rule {
        Replace bigram (X,Y) with X
      }
    }
  }
  Measure Grammar Model Length  $O(N \cdot (1+S))$ 
}

```

**Figure 6:** Pseudo-code of a single step of the application of the CreateOptionalNT operator.

As a first step, all possible bigrams whose first symbol has been created by the *CreateNT* operator must be identified, which requires an iteration over all symbols in all rule bodies. The complexity of this action is  $O(N \cdot S)$ , assuming that the process of storing the symbols is of constant complexity.

The number of the produced bigrams (represented by  $K$ ) can be at most  $K = N \cdot S$ .

As a second step, the CreateOptionalNT operator must be applied over all bigrams. For each bigram we have to identify the set  $S_H$ . This is an iterative process which can perform at most  $N-1$  steps (if all rule heads except the grammar start symbol have been appended in  $S_H$ ). Since all rules must be examined during each iteration, the total complexity of identifying  $S_H$  equals to  $(N-1) \cdot N$ . The total number of elements in  $S_H$  can be at most  $L = N-1$ .

Finally, the third step is to apply the CreateOptionalNT operator and measure the model length of the produced grammar. Applying the operator requires an iteration over all symbols in  $S_H$ : for each symbol in  $S_H$  all rule bodies from all rules in the grammar must be examined in order to eliminate the symbol “Y”. This process has a complexity of  $L \cdot N \cdot S$ , assuming that the process of replacing a bigram with a symbol is of constant complexity. Considering all steps, the total complexity of the mode for the CreateOptionalNT operator ( $C_{CreateOptionalNT}$ ) is cubic:

$$\begin{aligned}
C_{CreateOptionalNT} &= O(N \cdot S + K \cdot N(N + L \cdot S + S)) = O(N \cdot S + N^2 \cdot S \cdot (N + (N-1) \cdot S + S)) = \\
&O(N^3 \cdot S + N^2 \cdot (N-1) \cdot S^2 + N^2 \cdot S^2 + N \cdot S) \approx O(2 \cdot N^3 + N^2)
\end{aligned}$$

### 3.3.3 The effect of “Create Optional NT” on Grammar Description Length

As was the case with the MergeNT operator, the initial *GDL* of  $G$  (before the operator is applied) can be calculated as:

$$GDL_{In} = (A_{NT} + A_R - A_S + 2) \cdot \log(A_{UNT} + 1) + A_T \cdot \log(A_{UT}).$$

Similarly, the *GDL* after the operator has been applied can be calculated as:

$$GDL_{Fin} = (A_{NT} + A_R - A_S + 7) \cdot \log(A_{UNT} + 1) + A_T \cdot \log(A_{UT}) - E_1 - E_2$$

The term  $E_1$  in the above equation represents the expected reduction in *GDL* that can occur if the application of this operator leads to duplicate rules, which should be removed. Recalling the discussion of section 3.2.3, we have:

$$E_1 = \sum_{j \in \Omega_1} (L_j + 1) \cdot \log(A_{UNT} + 1) + \sum_{j \in \Omega_3} (L_j + 2) \cdot \log(A_{UNT} + 1)$$

where:

- $\Omega_1$ : the set of the rules from the start symbol subset that are eliminated from  $G$ .
- $\Omega_3$ : the set of the rules from the non-terminal subset that are eliminated from  $G$ .
- $L_j$ : the number of non-terminal symbols in the body of rule  $j$ .

The term  $E_2$  represents the expected reduction in *GDL* that occurs from the removal of the symbol “ $Y$ ”:

$$E_2 = N_Y \cdot \log(A_{UNT} + 1)$$

where  $N_Y$  is the number of removals of the symbol “ $Y$ ” from the grammar. As a result, the change  $\Delta_{GDL}$  of the *GDL* due to this operator is:

$$\Delta_{GDL} = 5 \cdot \log(A_{UNT} + 1) - N_Y \cdot \log(A_{UNT} + 1) - \left( \sum_{j \in \Omega_1} (L_j + 1) \cdot \log(A_{UNT}) + \sum_{j \in \Omega_3} (L_j + 2) \cdot \log(A_{UNT}) \right). \quad (7a)$$

The first term in equation 7a is a small positive constant, while the other two remaining factors are always negative (due to the minus signs). Due to the fact that the only positive quantity is only a small positive constant, even the elimination of five occurrences of the symbol “ $Y$ ” suffices to get a reduction in the *GDL* of the produced grammar. The reduction in *GDL* is larger as the number of the duplicate rules eliminated from the grammar and their length increases as well as the number of occurrences of the symbol “ $Y$ ” that are eliminated increases.

### 3.3.4 The effect of “Create Optional NT” on Derivations Description Length

Like the MergeNT operator, the CreateOptionalNT operator modifies *DDL* in two ways: by merging sets of rules and by eliminating rules from the grammar. Regarding the effect on the *DDL* due to the addition of the new rule that shares the same head with an existing rule, it is identical to the merge of two rule sets with the second set containing a single rule. Thus, if we set  $\beta = 0$  and  $F_Y = 1$  to equation 6d, the equation becomes equal to the following quantity:

$$\Delta C = \alpha \cdot \log \left( 1 + \frac{1}{F_X} \right) \quad (7b)$$

Finally, regarding the elimination of rules from the grammar due to the fact that they have become duplicated, the effect of the CreateOptionalNT operator is identical to the effect of the MergeNT operator, as described by equation 6e (with symbols “X” and “Z” of the MergeNT operator now identified as the symbols “X” and “Y” of the CreateOptionalNT operator).

### 3.3.5 Total contribution of “Create Optional NT” to the Model Description Length

The total contribution of the CreateOptionalNT operator to the model description length of a grammar can be calculated by combining equations 7a, 7b and 6e.

$$\begin{aligned}
 \Delta ML = & 5 \cdot \log(A_{UNT} + 1) - N_Y \cdot \log(A_{UNT} + 1) - \\
 & \left( \sum_{j \in \Omega_1} (L_j + 1) \cdot \log(A_{UNT}) + \sum_{j \in \Omega_3} (L_j + 2) \cdot \log(A_{UNT}) \right) + \\
 & \left( \sum_{j \in G_m} N_j^{CNT-X} \cdot F_j^S \right) \cdot \log \left( 1 + \frac{1}{F_{CNT-X}} \right) + \sum_{j \in G_{Fn}} \left( \sum_{k \in \Theta, j} M_k \right) \cdot F_j^S
 \end{aligned} \tag{7}$$

The first three terms of equation 7 correspond to the change in *GDL*. The fourth and fifth terms of equation 7 correspond to the change in *DDL*: The fourth term represents the merging of sets of rules and is always positive. The last term represents the elimination of duplicated rules and is always negative or zero, depending on the existence of duplicated rules.

## 4 Experimental Evaluation

In this section we evaluate the e-GRIDS algorithm experimentally, focusing on its performance on specific tasks and its scalability. Additionally, we examine the role of certain parameters of the algorithm, like the size of the beam. We use both sentences generated from artificial grammars and sentences from a large textual corpus in our studies. Artificial grammars let us evaluate specific characteristics of e-GRIDS, as we can control various aspects of a grammar, and they give us the ability to measure the correctness of the induced grammars. The large textual corpus, on the other hand, lets us examine the scalability of e-GRIDS to more complex grammatical domains.

### 4.1 Experiments on Artificial Grammars

#### Evaluation Metrics for Artificial Grammars

Evaluation in grammatical inference presents some peculiarities and common metrics that are used for supervised learning tasks, like recall and precision, are not directly applicable. Alternatively, in order to evaluate an inferred grammar we have to compare it against the “correct” grammar, so as to identify their similarity. However, even if the “correct” grammar is known, which is not the case in most real-world situations, the problem of determining whether two context-free grammars are equivalent is not an easy task: Given two context-free grammars  $G_1$  and  $G_2$ , there exists no algorithm that can determine whether  $G_1$  is more general than  $G_2$  (i.e.  $L(G_1) \supseteq L(G_2)$ ) or if  $L(G_1) \cap L(G_2) = \emptyset$ , where  $L(G)$  the language of grammar  $G$  [20, 13].

As a result, during evaluation we mainly focus on measuring three aspects of the inferred grammar [15]:

- errors of omission (failures to parse sentences generated by the “correct” grammar), which indicate that an overly specific grammar has been learned,
- errors of commission (failures of the “correct” grammar to parse sentences generated by the inferred grammar), which indicate that an overly general grammar has been learned, and
- ability of the inferred grammar to parse correctly sentences longer than the sentences used during training, which indicates the additional expressiveness of the learned grammar.

In experiments with artificial grammars, where the “correct” grammar is known, to estimate these figures we use the original (or “correct”) grammar  $G_O$  and the learned grammar  $G_L$  to generate a large number of sentences. Errors of omission can be estimated as the fraction of the number of sentences generated by  $G_O$  that are not parsed by  $G_L$  to the total number of sentences generated by  $G_O$ . Errors of commission can be estimated as the fraction of the number of sentences generated by  $G_L$  that are not parsed by  $G_O$  to the total number of sentences generated by  $G_L$ . Errors of omission and errors of commission measure the *overlap* of the two grammars. In the ideal case, both of these figures must be zero, indicating that all sentences generated by one grammar can be parsed by the other. If this is the case and a sufficiently large number of sentences has been generated, we can conclude that the two grammars significantly overlap. In order to estimate the third figure, example sentences must be generated from  $G_O$  that have greater length than the ones used for training. This figure can then be estimated as the fraction of the number of sentences that were successfully parsed by  $G_L$  to the total number of generated sentences.

#### Experimental Setting

The first set of experiments evaluates e-GRIDS on examples generated from simple recursive grammars and involve a small set of terminal symbols, as we are mainly interested in examining the ability of e-

GRIDS to infer recursive grammars. The two grammars that were used in our experiments are shown in Table 11. The first grammar (a) includes declarative sentences with arbitrarily long strings of adjectives and both transitive and intransitive verbs, but no relative clauses, prepositional phrases, adverbs or inflections. The second grammar (b) contains declarative sentences with arbitrary embedded relative clauses but has no adjectives, adverbs, prepositional phrases or inflections. Both grammars are associated with a small lexicon of terminal symbols.

(a)		(b)	
S	→ NP VP	S	→ NP VP
VP	→ VERBI	VP	→ VERB NP
VP	→ VERBT NP	NP	→ ART NOUN
NP	→ the NOUN	NP	→ ART NOUN RC
NP	→ the AP NOUN	RC	→ REL VP
AP	→ ADJ	VERB	→ saw
AP	→ ADJ AP	VERB	→ heard
VERBI	→ ate	NOUN	→ cat
VERBI	→ slept	NOUN	→ dog
VERBT	→ saw	NOUN	→ mouse
VERBT	→ heard	ART	→ a
NOUN	→ cat	ART	→ the
NOUN	→ dog	REL	→ that
ADJ	→ big		
ADJ	→ old		

Table 11: Two artificial grammars: grammar (a) includes arbitrary strings of adjectives, and grammar (b) supports arbitrary relative clauses [15].

Although these grammars are simplistic compared to natural language grammars, they both involve recursion and thus describe an infinite language. The first grammar uses a simple type of recursion, where adjectives are repeated before a noun. Once all adjectives are grouped into a single non-terminal by e-GRIDS, sequences of symbols belonging to this category usually provide enough information to detect recursion. On the other hand, the second grammar involves recursion over a relative clause: the relative clause must be first correctly identified before enough evidence to detect recursion appears. The main point of interest in both of these grammars is to examine the ability of e-GRIDS to generalise correctly, by inferring grammars that involve recursion.

In the experiments that we have conducted, we have generated a large number of sentences top-down from grammars (a) and (b). For each grammar, a large number of example sentences (more than 10000) was generated, using a uniform distribution to select rules randomly<sup>1</sup> when expanding ambiguous non-terminals. The generated example sentences were randomly shuffled. For each grammar we defined an arbitrary maximum length  $L_{max}$  for the training examples and the test examples. The resulting set was used for evaluation according to the two first figures, i.e. errors of omission and commission. As we wanted to study whether e-GRIDS can produce recursive grammars, we wanted also to evaluate the learned grammars on example sentences that were longer than the longest ones used for training. Therefore, a second test set was created, containing example sentences with lengths greater than  $L_{max}$  but lower than a second arbitrary maximum length. This second set is the set that was used in order to calculate the third figure, i.e. the ability to parse sentences longer than the ones used for training. All sets were populated by randomly selecting example sentences from the generated sentences. Special care

<sup>1</sup>All random selections in our experiments were based on a uniform distribution.

was taken in order to ensure that the same sentence did not appear both in the training and the test sets.

As we are also interested in the learning rate, we have varied the size of the training and test sets presented to e-GRIDS. In order to gain an unbiased estimate of the performance of e-GRIDS on unseen data, each experiment was repeated ten times. For every training set size, ten equally-sized training sets of the desired size and a test set of the same size plus 1000 sentences were created. Half of the example sentences in the test set were longer than  $L_{max}$ . Each one of the ten training sets was used to train the learning algorithm and the performance of the inferred grammar was evaluated on the common test set. The average over the ten runs is reported here as the final evaluation measure along with the standard deviation. Note that an example sentence is considered successfully parsed once at least one parse have been found given a grammar. No attention has been paid to the quality of the parse trees.

### Experiment 1: Evaluation on small grammars

As a first experiment, we have evaluated the ability of e-GRIDS to infer recursive grammars by using the two grammars presented in Table 11. Regarding grammar (a) from Table 11, a set of 22826 sentences was generated. This set was split into two subsets. The first subset (A) contained sentences with length up to 15 words (21183 sentences), while the second subset (B) contained sentences with length between 16 and 20 words (1643 sentences). Beam search was used with a beam size of 3. The experiment was conducted over various training set sizes, from 10 to 700 sentences. For each training set size, ten training sets were created. Each training set contained a fixed number of unique sentences that were randomly chosen from subset (A). The ten different sets contained unique sentences, as no sentence from any one training set was shared with any of the other nine training sets. From these training sets, ten grammars were inferred and were evaluated on the common test set, originating from subset (B) and containing roughly 1000 sentences more than each training set. In figures<sup>2</sup> 3, 4 and 5, the averages are presented together with the error bars, measuring the standard deviation.

Figure 3 presents the probability of the learned grammar to parse a valid sentence that has length up to the maximum length of the examples used for training. Figure 4 presents the probability of parsing a valid sentence that has length greater than any of the examples seen by e-GRIDS during training. Finally, Figure 5 presents the probability of generating a valid sentence. From the two first curves we can conclude that e-GRIDS reaches an acceptable performance (i.e.  $\geq 0.9$ ) without requiring more than 350 examples. After 350 examples, the inferred grammars are general enough in order to recognise a large proportion of the unseen sentences, even if the sentences are longer than the sentences observed by e-GRIDS during training. Both curves of Figure 3 and Figure 4 represent the ability of e-GRIDS to generalise, while the curve of Figure 4 additionally provides an indication of the ability of e-GRIDS to learn recursion. The similarity of the two curves shows that e-GRIDS succeeded in finding the recursion over adjectives implied by the training sets.

Another interesting observation is the relatively large error bars, which are due to the “quantized” behaviour of the algorithm. The inferred grammars either generalised enough to recognise the test set (including also recursive rules) or were overly specific and did not generalise at all to unseen sentences. For example for training set size of 250, e-GRIDS inferred nine grammars that were able to recognise all the sentences in the test set and one grammar that failed completely on the same test set.

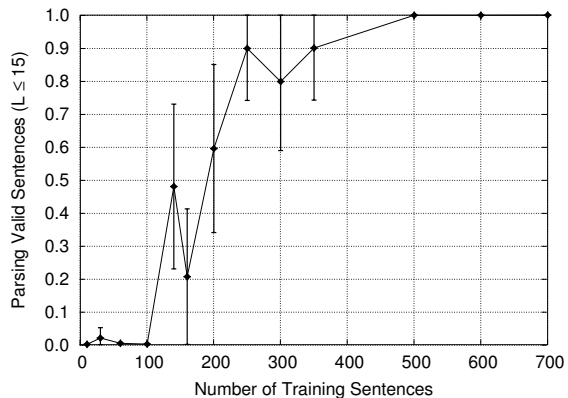


Figure 3: Probability of parsing a valid sentence of length up to 15 words. (1-errors of omission)

<sup>2</sup>Results for all experiments are expressed in (1-errors of omission) and (1-errors of comission), which were defined in section 4.1

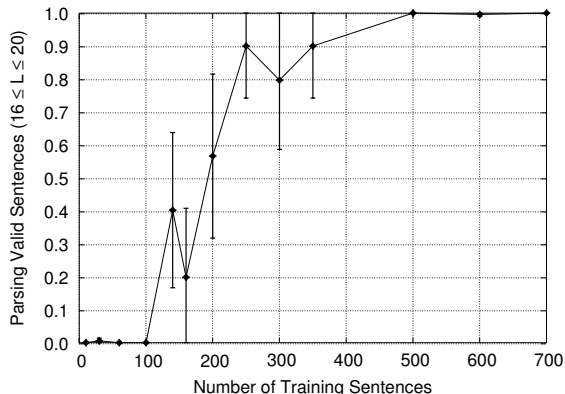


Figure 4: Probability of parsing a valid sentence of length between 16 and 20 words. (1-errors of omission)

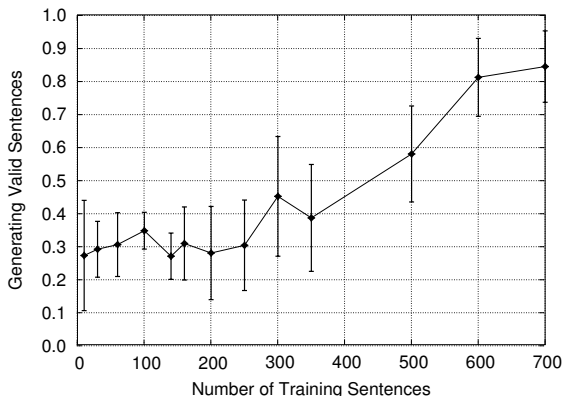


Figure 5: Probability of generating a valid sentence. (1-errors of commission)

The curve in Figure 5 clearly shows that e-GRIDS overgeneralises when trained with small training sets. Even at the size of 700, the inferred grammars have a probability 0.15 of generating ungrammatical sentences. The main reason behind this inefficiency is in the way e-GRIDS selects to categorise terminals. For example, consider the case in which e-GRIDS decides to classify a noun and an adjective under the same non-terminal symbol. This wrong categorisation may never create problems in recognising unseen sentences but it will lead to the generation of many ungrammatical sentences. The number of available examples is not sufficient for ten-fold validation experiments with training set sizes greater than 700 under the same experimental setting. However, single train – test experiments show that the probability of generating a legal sentence reaches 1.0 and stabilises for training set sizes greater than 700.

Regarding the second grammar (grammar (b) from Table 11), the experimental setting is exactly the same as the one for grammar (a). From this grammar, a set of 11500 sentences was generated. This set was again split into two subsets. The first subset contained sentences with length up to 15 words (8387 sentences), while the second subset contained sentences with length between 16 and 20 words (3113 sentences). The beam size was again 3. The experiment was conducted over various training set sizes (from 10 to 50 training examples) and the results showed that even for a small number of training examples (20 and above) e-GRIDS does no mistakes. In other words, all three evaluation measures used here take the value 1.0.

## Experiment 2: Varying the Beam Size

In a second experiment, we investigated the role of the beam size in the search process. The beam in e-GRIDS is used in order to store at any point of the learning process the  $B$  most prominent grammars. These are the grammars that will be further generalised by the three operators. In general, one expects that the performance will increase as the size of the beam increases and indeed this is what we observed. We have conducted two different tests, both of them based on the grammar (a), as grammar (b) seems to be easily learnable even for  $B = 3$ . In the first test the size of the beam was set to  $B = 1$ , i.e. no beam was used, while in the second test the beam size was set to  $B = 10$ . The results of the experiment with  $B = 1$  were very close to the results obtained with  $B = 3$ , and are therefore not repeated here.

The results of the second test ( $B = 10$ ) are shown in figures 6, 7 and 8. Comparing these graphs to those in figures 3, 4 and 5, we can conclude that the performance of e-GRIDS has improved. Performance according to all three measures reaches 1.0 with training sets of 600 sentences. Of course, the increased

beam size has a penalty on the required learning time: e-GRIDS requires more than 10 minutes to converge to a final grammar<sup>3</sup> when trained with a training set of size 700, while less than 2 minutes are required when the beam size is 3.

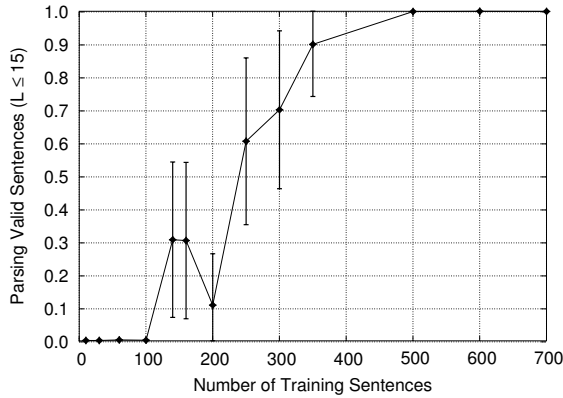


Figure 6: Probability of parsing a valid sentence of length up to 15 words. ( $B = 10$ )

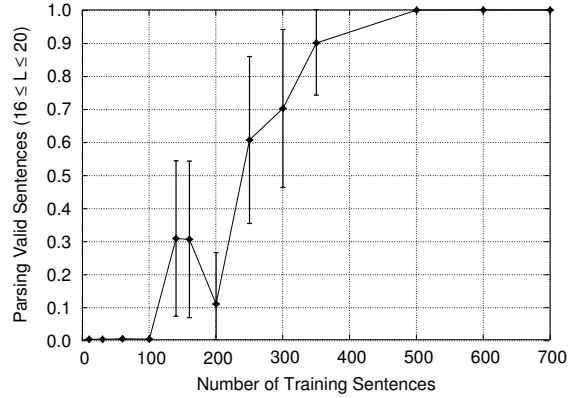


Figure 7: Probability of parsing a valid sentence of length between 16 and 20 words. ( $B = 10$ )

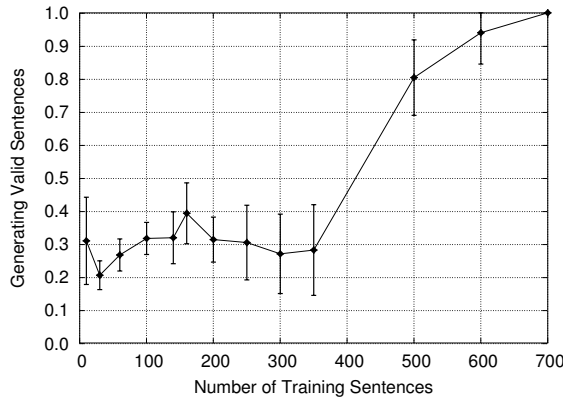


Figure 8: Probability of generating a valid sentence. ( $B = 10$ )

### Experiment 3: The balanced parenthesis language

Both grammars (a) and (b) in Table 11 can be converted to equivalent regular grammars. However, since e-GRIDS is able to infer context free grammars, it would be interesting to study its learning behaviour on a context free language. As an evaluation language we have chosen the Dyck language with  $k = 1$ :

$$S \rightarrow S S | [S] \in \quad (8)$$

As the Dyck language with  $k = 1$  cannot be used to generate a large number of example sentences if we restrict the maximum sentence length, we have performed a ten-fold cross validation. Similar to the procedure that we have followed in the previous experiments, subset (A) is split into ten subsets of

<sup>3</sup>On a PC with an AMD Athlon/1400 MHz processor running RedHat Linux.

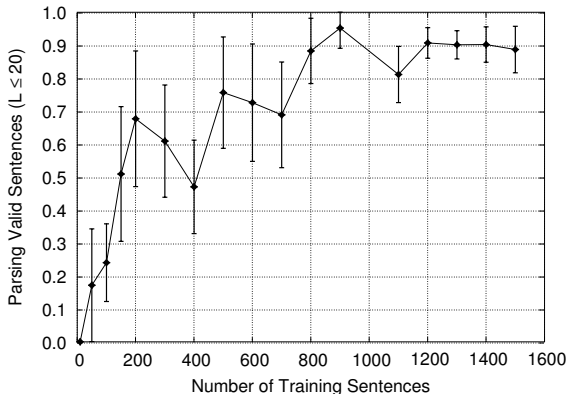


Figure 9: Probability of parsing a valid sentence of length up to 20 words. (1-errors of omission)

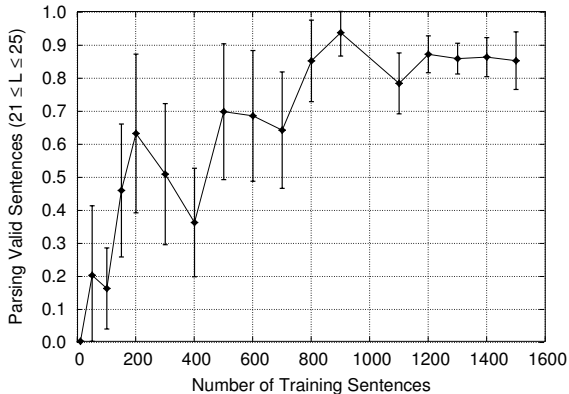


Figure 10: Probability of parsing a valid sentence of length between 21 and 25 words. (1-errors of omission)

equal size and one test set is created from subset (B). The experiment is repeated 10 times: each time 9 subsets of (A) are used for training e-GRIDS and the learned grammar is evaluated on the 10<sup>th</sup> unused set, augmented with the test set created from subset (B). Figures 9 and 10 show the results for this experiment. The figure for (1-errors of commission) is not presented as the performance of the algorithm according to this measure was 1.0 even for very small training set sizes, i.e. the learned grammar did not generate ungrammatical sentences.

Regarding the figures for parsing valid sentences either of length up to 20 tokens<sup>4</sup> (Figure 9) or of length from 21 to 25 (Figure 10), e-GRIDS approaches 0.95 with a training set size of 900 example sentences and remains around 0.90 for greater example set sizes. The reason behind this behaviour is that e-GRIDS converges to grammars that are less general than the target Dyck grammar in some of the 10 folds. Another interesting point is that unlike the previous experiments, e-GRIDS doesn't converge to the target Dyck grammar as shown in equation 8, even when reaching performance of 1.0 in all three figures. The learned grammars were more complex in the sense that a greater number of rules and non-terminal symbols were involved than in the correct grammar. Nevertheless, the learned grammars encoded the two main phenomena, the recursion and centre embedding of the Dyck language.

## 4.2 Experiments with a Large Textual Corpus

Although experiments with artificially generated sentences allowed the study of various features of e-GRIDS, experiments with sentences from linguistic corpora are required in order to examine e-GRIDS' robustness to real-world problems, as well as its scalability to more complex grammatical domains. However there are many difficulties associated with experimental evaluation on real corpora. Usually the "perfect" grammar is not available and as a result it is not possible to compare the learned grammars with it. Furthermore, since the perfect grammar is not known, no additional sentences can be generated in order to evaluate its overlap with a learned grammar. Thus, the required sentences need to be substituted by additional unseen sentences from the corpus, if the corpus is large enough. However, this does not solve the problem of measuring the overlap, as we cannot expect sentences generated from the learned grammars to exist in the corpus. The fact that generated sentences from learned grammars are not in the corpus does not necessarily mean that they are ungrammatical.

<sup>4</sup>A token is either a single left parenthesis "(" or a right one ")".

Another difficulty associated with these experiments is the fact that the set of words used in the sentences is not closed. Consider for example the case where a grammar is inferred from a small number of sentences, consisting of a limited set of words. The inferred grammar will never be able to parse sentences that contain words outside this limited set, as the words will not be represented by terminal symbols in the grammar. This shortcoming is hard to overcome. An experimental compromise is to specifically construct the training and test sets so that they contain the same words in their sentences. An easier solution is to insert an abstraction layer above the individual words that map words to a fixed set of symbols. For example, such an abstraction is to use the part of speech (POS) instead of the actual words.

The insertion of an abstraction layer may solve the problem of the open word set but it also leads to a new one: the abstraction layer is itself a *generalisation* over the original sentences. Consider for example the case where a single sentence is converted into a grammar. If each word is substituted by a POS category, the resulting grammar from this sentence will be able to recognise not only the sentence from which it originated but also any other sentence that contains the same POS sequence. The level of generalisation depends on the morphological details represented by the word categories. For example, a classification based only on the POS category of the words corresponds to a significant generalisation step. On the other hand, if the classification is augmented with additional information like gender, number and person, the level of generalisation is lower, as the number of categories is larger, with each category corresponding to a smaller number of words. A high degree of generalisation can have a negative impact on the probability of generating valid sentences using the inferred grammars. Even the training sentences themselves, when viewed as an initial grammar, may generate ungrammatical sentences. Things can get worse if words are misclassified into the abstraction categories by an automated system, e.g. a POS tagger. If POS categories are used as an abstraction layer, misclassifications are not rare as identical words can belong in different POS categories that can be difficult to disambiguate. As e-GRIDS only generalises an initial grammar extracted from the training sentences, it cannot eliminate these errors, converging to a final grammar that may generate many ungrammatical sentences.

Despite the problems associated with the use of an abstraction layer, we have chosen to use this paradigm for applying e-GRIDS to sentences from linguistic corpora. The reason we have not followed the paradigm of creating training and test sets that share the same words is mainly due to its unpredictable nature. Furthermore, the use of a very large number of terminal symbols would lead to an equally large subset of terminal rules in the initial grammar. This would result in an excessive increase in the grammar description length in comparison to the derivation length, which would prohibit the operators from improving the original grammar.

The corpus used in the experiment was a part of the SEMCORE corpus. The main purpose of this experiment was to evaluate the ability of e-GRIDS to handle large training sets and to converge to a final grammar in a reasonable amount of time. The time required by e-GRIDS to complete the task is shown in Figure 11. The graph displays the time required to complete the task by two versions of e-GRIDS: the first version uses the results of the theoretical analysis, as presented in sections 3.1.4 and 3.2.4, while the second does not use this analysis, resorting to the way the original GRIDS searched the space of possible grammars. The size of the beam was set to 1. As we can see from the graph, the time required by the optimised e-GRIDS is significantly lower than the time required by the simple GRIDS-like variant. This is very important, as it lets e-GRIDS be trained with significantly larger example sets when compared to GRIDS, converging to a final grammar within a reasonable amount of time.

It should also be noted that this optimisation is based on a theoretical analysis where no compromises have been made regarding the accuracy of the inferred grammars with respect to GRIDS. Further significant improvements are possible by tolerating a small degree of inaccuracy. For example, a significant amount of processing time is spent during the end of the training process, where e-GRIDS generates many grammars that are very similar to each other, with insignificant differences in the description length, in order to select the absolutely “best” one of them that will become the final grammar. By stopping

the search when the marginal gain in description length falls below a certain threshold, one can reduce processing time significantly. Additional optimisation is possible by calculating only the most dominant components of the equations that resulted from the theoretical analysis.

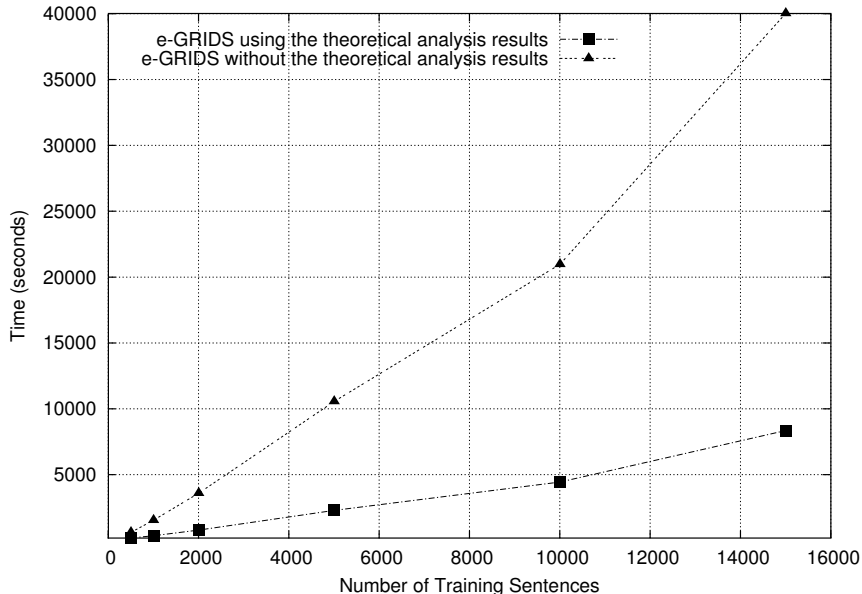


Figure 11: Time required by e-GRIDS, with and without the use of the theoretical analysis results.

The evaluation of the learning performance of e-GRIDS in this experiment is not straightforward as the learning task is not well-defined. However, in order to get an indication of the ability of e-GRIDS to generalise from large training sets, we have also performed a simple check. After training e-GRIDS with 2000 example sentences, we randomly selected 200 unseen example sentences and we counted how many unseen sentences could be parsed by both the learned grammar, as well as by the training example set converted to an (initial) grammar. The initial grammar was able to parse 25 sentences, while the learned grammar was able to parse 4 more sentences, with a total score of 29 correctly parsed sentences. These figures, provide an initial indication of e-GRIDS’ ability to converge to more general grammars than the base case. Nevertheless, further experimentation in a more controlled experiment is needed in order to prove the value of e-GRIDS in learning natural language CFGs.

## 5 Related Work

e-GRIDS shares some of its central features with earlier work in grammatical inference. We have already mentioned that e-GRIDS originates from GRIDS [15] which is in turn based on SNPR [34, 35]. The SNPR system uses similar learning operators as GRIDS and e-GRIDS, with the exception that possibly more than two symbols can be handled by an operation. SNPR is also biased towards “simple” grammars, as it uses *MDL* for scoring and selecting the most plausible grammars. Furthermore, like e-GRIDS, SNPR uses heuristics for avoiding the generation of all grammars in each iteration. However, the choice of these heuristics is based solely on intuition and may not allow SNPR to converge to the optimal grammar according to the *MDL*.

Although the majority of the work in grammatical inference focuses on regular grammars, a small number of algorithms exist that infer context-free grammars. Stolcke and Omohundro [30, 29] have

presented an approach which infers probabilistic context-free grammars. Using a Bayesian framework, their system employs similar learning operators as e-GRIDS and tries to find a grammar with maximal posterior probability given the training example set. This criterion is essentially equivalent to the *MDL*, employed by e-GRIDS. In [27] an algorithm for inducing context-free grammars from positive and negative structured examples is presented. A structured example is simply an example with some parentheses inserted to indicate the shape of the derivation tree of the grammar (structural information). The learning algorithm employs a genetic search and the CKY algorithm [13] for converging to a final grammar. An efficient successor of this algorithm can be found in [16]. A more recent version of this algorithm [28] operates on partially structured examples instead of complete ones and uses a tabular representation, leading to a more flexible and applicable algorithm, compared to its predecessor. The algorithm has been successfully applied to various simple languages as well as to DNA sequence modelling. *Synapse* [18] is another algorithm that is based on CKY. *Synapse* learns incrementally from positive and negative examples, while preliminary results show that it is able to infer both ambiguous and unambiguous context-free grammars for simple languages in reasonable time.

Various attempts have also been made to apply grammatical inference algorithms to natural language learning tasks. In [19] a stochastic variant of Sakakibara’s algorithm is evaluated on part of the Penn Treebank [17] with the task of learning the syntax of the language from positive structured examples. Freitag [9] applies three grammatical inference algorithms (ALERGIA [4], ECGI [26] and a Bayesian one) to the task of information extraction from seminar announcements. In the same work, a special learning technique is proposed that learns “alphabet transducers”, components that re-encode text in an abstract representation, more suitable for applying grammatical inference than plain text. In [1] an algorithm for inferring context-free grammars that try to locate user-specified fields in structured documents is presented. The algorithm works by creating a prefix tree automaton from all instances of the user specified fields in the documents. These automata are generalised with the help of heuristics, converted to regular expressions and combined to form the final context-free grammar. In [11] ALERGIA, along with the new algorithm WIL, are applied to the task of wrapper induction for web pages. The ABL algorithm [38] takes as input a corpus of flat (unstructured) sentences and returns a corpus of labelled and bracketed sentences. In [37] the ABL and the EMILE algorithms are compared on the ATIS corpus. Clark [5] presents a new algorithm that uses content distribution clustering for inducing stochastic context-free grammars from tagged text. This algorithm is evaluated on the task of inferring phrase structured grammars from the British National Corpus, as well as from the ATIS corpus.

The vast majority of the algorithms applied to natural language learning tasks infer grammars solely from positive examples, as natural language learning is a characteristic task where negative examples are rarely available. Another interesting observation regarding the application of grammatical inference algorithms to natural language learning, is that many of these algorithms exhibit a very long convergence time in practice. It seems that except from the absence of negative examples, natural language learning places additional strains to the algorithms as it requires learning from languages with a large alphabet, which are also relatively complex, necessitating the examination of a relative large set of examples. As a result, the ability of a grammatical inference algorithm to converge in reasonable time when trained on large example sets is not to be underestimated in this domain.

## 6 Conclusions

In this paper, we presented the e-GRIDS algorithm for inducing context-free grammars solely from positive evidence. e-GRIDS utilises a heuristic based on minimum description length to avoid overgeneralisation, a consequence of the absence of negative evidence. One of its main advantages is its computational efficiency which facilitates its scalability to large example sets. The dynamic behaviour of the search operators has been analysed theoretically, leading to the optimisation of the inference process, by not

requiring the generation of all grammars in each iteration. The performance of e-GRIDS was evaluated on artificially generated example sets, as well as on a large textual corpus.

Regarding the learning performance of e-GRIDS, experiments have been conducted with the help of artificially generated examples. Our results have shown that e-GRIDS is able to infer grammars that perform well, based on relatively small sets of training examples. The incorporated minimum description length heuristic appears to help e-GRIDS to avoid overgeneralisation, at least for the simple artificial languages that we examined. e-GRIDS seems to be able to generalise correctly, by inferring grammars that successfully model the provided training example sets and at the same time do not generate ungrammatical sentences. Another interesting feature of e-GRIDS is the ability to infer grammars able to recognise longer example sentences than the ones presented to the algorithm during training, as e-GRIDS is able to infer recursive grammars modelling simple recursive cases, i.e. when some word classes or phrases tend to repeat.

More crucial for practical applications is the ability of the algorithm to scale to large example sets. Regarding the computational efficiency of e-GRIDS, results are very satisfactory as e-GRIDS is able to handle large example sets in significantly reduced amounts of time when compared to a simple GRIDS-like variant. e-GRIDS utilises the results of a theoretical analysis of the dynamic behaviour of its learning operators, where no compromises have been made regarding the accuracy of the inferred grammars with respect to its predecessor, the GRIDS algorithm. If some small accuracy compromises were tolerated, e-GRIDS could be made even more efficient.

The work presented here has paved the way for further improvements and extensions to the e-GRIDS algorithm. The most interesting extensions are those required to tackle complex linguistic tasks. In this direction, the handling of attribute grammars is particularly interesting. Learning attribute grammars is essential for various tasks where attributes are associated to terminal features. For example, in named-entity recognition, each word is usually associated with additional information beyond its part of speech, such as capitalisation information or information related to whether the word has been identified as part of a known named-entity contained in a gazetteer. This information is essential for the task of recognising named entities. Thus, introducing attribute support in e-GRIDS will offer an ideal vehicle for modelling linguistic tasks.

Furthermore, e-GRIDS could be evaluated further and compared to other existing algorithms. An interesting task for future work is to compare the computational efficiency of e-GRIDS with other algorithms that have been applied to tasks involving complex languages with large alphabets (e.g. various NLP tasks). Such an evaluation will also enable the comparison of the accuracy of the grammars inferred by e-GRIDS to those inferred by other algorithms, provided that the same experimental setting can be reconstructed.

Another interesting aspect that can be examined as future work is the ability of e-GRIDS to learn in an incremental manner. The current implementation of e-GRIDS offers the ability to load a grammar inferred by e-GRIDS at an earlier time and generalise it with respect to a new set of examples. It would be interesting to examine the learning behaviour of e-GRIDS when learning in incremental mode in comparison to when learning in batch.

An additional facility provided by e-GRIDS is the definition of alternative strategies for applying the learning operators. For example, e-GRIDS can operate in a way where the three operators are interleaved, instead of each operator being applied continuously until it cannot lead to further improvement. In this paper we have done significant work in understanding the effect of each operator separately to the induced grammar. Future work could focus on the effect of alternative strategies for using the three operators.

Concluding, the work presented here has resulted in a new algorithm that alleviates many of the shortcomings of its predecessors, with special attention given to its robustness and computational efficiency. We believe that e-GRIDS will be useful in modelling various subparts of natural languages and identifying these subparts in texts, a task that cannot be easily modelled by other machine learning approaches, at least those that expect fixed-length vectors as input. Even if some machine learning approaches can be

applied to such tasks, grammars will still have an advantage as they are the most “natural” and straightforward way of locating strings in texts. Interesting tasks that fit this description include noun phrase chunking, named-entity recognition and information extraction.

## References

- [1] H. Ahonen, H. Mannila, “Forming grammars for structured documents: an application of grammatical inference”, *Proceedings of International Colloquium on Grammatical Inference (ICGI-94)*, Carrasco R. and Oncina J. (eds), Lecture Notes in Artificial Intelligence 862, Springer – Verlag, pp. 153–167, 1994.
- [2] D. Angluin, M. Kharitonov, “When won’t membership queries help?”, *Proceedings of the 24<sup>th</sup> ACM Symposium on Theory of Computing*, ACM Press, pp. 444–454, New York, 1991.
- [3] D. Angluin, “Inference of reversible languages”, *Journal of ACM*, vol. 29, pp. 741–765, 1982.
- [4] R. Carrasco, J. Oncina, “Learning stochastic regular grammars by means of a state merging method”, *Proceedings of International Colloquium on Grammatical Inference (ICGI-94)*, Carrasco R. and Oncina J. (eds), Lecture Notes in Artificial Intelligence 862, Springer – Verlag, pp. 139–150, 1994.
- [5] A. Clark, “Unsupervised induction of stochastic context-free grammars using distributional clustering”, *Proceedings of the Fifth Conference on Natural Language Learning*, 2001.
- [6] F. Denis, “Learning Regular Languages from Simple Positive Examples”, *Journal of Machine Learning*, vol. 44, pp. 37–66, July 2001.
- [7] F. Denis, “PAC learning from positive statistical queries”, *M. M. Richter, C. H. Smith, R. Wiehagen and T. Zeugmann (eds), Proceedings of the 9<sup>th</sup> International Conference on Algorithmic Learning Theory (ALT-98)*, Berlin Springer Vol. 1501 of LNAI, pp 112–126, 1998.
- [8] J. D. Emerald, K. G. Subramanian, D. G. Thomas, “Learning Code regular and Code linear languages”, *Proceedings of International Colloquium on Grammatical Inference (ICGI-96)*, Lecture Notes in Artificial Intelligence 1147, Springer – Verlag, pp. 211–221, 1996.
- [9] D. Freitag, “Using grammatical inference to improve precision in information extraction”, *Working Papers of the ICML-97 Workshop on Automata Induction, Grammatical Inference, and Language Acquisition*, 1997. (<http://citeseer.nj.nec.com/freitag97using.html>)
- [10] P. García, E. Vidal, “Inference of K-testable languages in the strict sense and applications to syntactic pattern recognition”, *Journal of IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12(9), pp. 920–925, 1990.
- [11] T. Goan, N. Benson, O. Etzioni, “A grammar inference algorithm for the World Wide Web”, *Proceedings of AAAI Spring Symposium on Machine Learning in Information Access*, 1996.
- [12] E. M. Gold, “Language identification in the limit”, *Information Control*, vol. 10, pp. 447–474, 1967.
- [13] J. Hopcroft, J. Ullman, *Introduction to Automata Theory, Languages and Computation*. Addison – Wesley, 1979.
- [14] T. Koshiba, E. Makinen, Y. Takada, “Inferring pure context-free languages from positive data”, *Technical report A-1997-14*, Department of Computer Science, University of Tampere, 1997.
- [15] P. Langley, S. Stromsten, “Learning Context-Free Grammars with a Simplicity Bias”, *Proceedings of the Eleventh European Conference on Machine Learning (ECML 2000)*, Lecture Notes in Artificial Intelligence 1810, Springer – Verlag, pp. 220–228, Barcelona, Spain, 2000.
- [16] F. Mäkinen, “On the structural grammatical inference problem for some classes of context-free grammars”, *Information Processing Letters*, 42, pp. 193–199, 1992.
- [17] M. P. Marcus, B. Santorini, M. A. Marcinkiewicz, “Building a Large Annotated Corpus of English: The Penn Treebank”, *Computational Linguistics*, 19(2), pp. 313–330, 1993.

- [18] K. Nakamura, T. Ishiwata, “Synthesizing Context Free Grammars from Simple Strings Based on Inductive CYK Algorithm”, *Proceedings of the Fifth International Colloquium on Grammatical Inference (ICGI 2000)*, *Grammatical Inference: Algorithms and Applications*, Oliveira A. (ed.), Portugal, 2000. Springer.
- [19] F. Nevado, J. Sánchez, J. Benedí, “Combination of Estimation Algorithms and Grammatical Inference Techniques to Learn Stochastic Context-Free Grammars”, *Proceedings of the Fifth International Colloquium on Grammatical Inference (ICGI 2000)*, *Grammatical Inference: Algorithms and Applications*, Oliveira A. (ed.), Portugal, 2000. Springer.
- [20] R. Parekh, V. Honavar, “Grammar Inference, Automata Induction, and Language Acquisition”, R. Dale, H. Moisl, and H. Somers (eds.), *Handbook of Natural Language Processing*, chapter 29, pp. 727–764, Marcel Dekker Inc., 2000.
- [21] G. Petasis, G. Paliouras, V. Karkaletsis, C. Halatsis, C. D. Spyropoulos, “e-GRIDS: Computationally Efficient Grammatical Inference from Positive Examples”. *Technical Report 2004/1*, NCSR “Demokritos”, Athens, 2004. (<http://www.iit.demokritos.gr/~petasis>)
- [22] J. R. Quinlan, *C4.5 Programs for machine learning*, Morgan-Kaufmann, San Mateo, CA, 1993.
- [23] J. Rissanen, *Stochastic Complexity in Statistical Inquiry*, World Scientific Publishing Co, Singapore, 1989.
- [24] D. Ron, Y. Singer, N. Tishby, “On the Learnability and Usage of Acyclic Probabilistic Finite Automata”, *Proceedings of Computational Learning Theory (COLT 1995)*, pp. 31–40, 1995.
- [25] H. Rulot, E. Vidal, “Modelling (sub)string-length-based constraints through grammatical inference methods”, Devijer and Kittler (eds.), Springer – Verlag, 1987.
- [26] H. Rulot, E. Vidal, “An Efficient Algorithm for the Inference of Circuit-Free Automata”, Ferrate G. A. (ed.), *Syntactic and Structural Pattern Recognition*, Berlin: Springer – Verlag, 1988.
- [27] Y. Sakakibara, “Efficient learning of context-free grammars from positive structural examples”, *Information and Computation*, 97, pp. 23–60, 1992.
- [28] Y. Sakakibara, H. Muramatsu, “Learning Context-Free Grammars from Partially Structured Examples”, *Proceedings of the Fifth International Colloquium on Grammatical Inference (ICGI 2000)*, *Grammatical Inference: Algorithms and Applications*, Oliveira A. (ed.), Portugal, 2000. Springer.
- [29] A. Stolcke, “Bayesian Learning of Probabilistic Language Models”, *PhD Thesis, University of California at Berkley*, 1994.
- [30] A. Stolcke, S. Omohundro, “Inducing Probabilistic Grammars by Bayesian Model Merging”, *Proceedings of International Colloquium on Grammatical Inference (ICGI-94)*, Lecture Notes in Artificial Intelligence 862, Springer – Verlag, pp. 106–118, 1994.
- [31] N. Tanida, T. Yokomori, “Inductive Inference of Monogenic Pure Context-free languages”, Lecture Notes in Artificial Intelligence 872, Springer – Verlag, pp. 560–573, 1994.
- [32] L. Valiant, “A theory of the learnable”, *Communications of ACM*, 2711, pp 1134–1142, 1984.
- [33] K. VanLehn, W. Ball, “A version space approach to learning context-free grammars”, *Journal of Machine Learning*, vol. 2, pp. 39–74, 1987.
- [34] G. Wolff, “Grammar Discovery as data compression”, *Proceedings of the AISB/GI Conference on Artificial Intelligence*, pp. 375–379, Hamburg, West Germany, 1978.
- [35] G. Wolff, “Language Acquisition, Data Compression and Generalisation”, *Language and Communication*, 2, pp. 57–89, 1982.
- [36] T. Yokomori, “On Polynomial-Time Learnability in the Limit of Strictly Deterministic Automata”, *Journal of Machine Learning*, vol. 19, pp. 153–179, 1995.
- [37] M. van Zaanen, P. Adriaans, “Comparing two unsupervised grammar induction systems: Alignment-based learning vs. EMILE”, *Research Report Series 2001.05*, School of Computing, University of Leeds, March, 2001.

- [38] M. van Zaanen, “ABL: Alignment-based learning”, *Proceedings of the 18<sup>th</sup> International Conference on Computational Linguistics (COLING-2000)*, pp. 961–967, 2000.