applied sciences

Article

Split-Based Algorithm for Weighted Context-Free Grammar Induction

Mateusz Gabor [1], **Wojciech Wieczorek** [2] **and Olgierd Unold** [3,*]

[1] Department of Field Theory, Electronic Circuits and Optoelectronics, Wroclaw University of Science and Technology, 50-370 Wroclaw, Poland; mateusz.gabor@pwr.edu.pl

[2] Department of Computer Science and Automatics, University of Bielsko-Biala, 43-309 Bielsko-Biala, Poland; wwieczorek@ath.bielsko.pl

[3] Department of Computer Engineering, Wroclaw University of Science and Technology, 50-370 Wroclaw, Poland

* Correspondence: olgierd.unold@pwr.edu.pl

Abstract: The split-based method in a weighted context-free grammar (WCFG) induction was formalised and verified on a comprehensive set of context-free languages. WCFG is learned using a novel grammatical inference method. The proposed method learns WCFG from both positive and negative samples, whereas the weights of rules are estimated using a novel Inside–Outside Contrastive Estimation algorithm. The results showed that our approach outperforms in terms of F1 scores of other state-of-the-art methods.

Keywords: grammar inference; weighted context-free grammar; split algorithm; unsupervised learning

Citation: Gabor, M.; Wieczorek, W.; Unold, O. Split-Based Algorithm for Weighted Context-Free Grammar Induction. *Appl. Sci.* **2021**, *11*, 1030. https://doi.org/10.3390/app11031030

Academic Editor: Akemi Galvez Tomida

Received: 16 November 2020

Accepted: 20 January 2021

Published: 24 January 2021

1. Introduction

The task of grammar or automata induction is a part of symbolic artificial intelligence [1] and is called grammatical inference or grammar induction [2]. Among different subtasks of this scientific field, learning (stochastic or more general weighted) context-free grammars (CFGs) from input data has been growing in importance, due to its practical implications such as natural language and biological sequences modelling.

Learning CFG is known to be a hard task and notable open questions are still open [2]. According to Gold's theorem [3], CFGs cannot be learned from positive examples only, but in 1969 Horning proved that for effective probabilistic/stochastic CFG (PCFG) induction no negative evidence is obligatory [4]. It is imperative to note that learning PCFG only from positive data leads to grammars, thereby making it difficult to discriminate negative sequences from the input data. To overcome these difficulties, we have recently proposed the novel algorithm for weighted CFG (WCFG) learning [5,6]. Weighted Grammar-based Classifier System (WGCS) is one of the few grammatical inference approaches learning both grammar structure (i.e., rules) and stochastic grammar parameters (i.e., weights of rules). Initially, the method was dedicated to learning crisp context-free grammar [7], and later, it was extended to weighted versions (including fuzzy one [8] or stochastic [9]).

WGCS is learned in an unsupervised manner from unannotated data such as, a structured corpus or treebank. There are some other unsupervised grammatical inference methods like ABL [10], EMILE [11], ADIOS [12], or LS [13]. However, none of these methods induces both structure and parameters of grammar.

The main contribution of this paper is to define and test a new version of WGCS approach, in which the split concept has been employed to reveal the grammar structure. Although the split was used for the first time in [6], its verification was rudimentary and limited due to the unrepresentative bioinformatics dataset. Subsequently, a new approach was formalised and tested over a comprehensive set of artificial CFG datasets, and its computational complexity was given. Moreover, the improved WGCS was compared with two state-of-the-art unsupervised methods—LS [13] and ADIOS [12]—dedicated to CFG learning. Additionally, the rule weight estimation algorithm was improved by mitigating unbalanced data bias.

The rest of the paper is organised as follows. Section 2 gives some details about our approach. In Section 3, we present a test environment and eventually the results are reported in Section 4. Section 5 concludes the paper.

2. Weighted Grammar-Based Classifier System

WGCS belongs to the family of learning classification systems [14] and is based on a previous version of [7] that only works on context-free grammars with no probabilities or weights. According to the idea of grammatical inference, WGCS system receives a set of tagged positive and negative sentences as an input to the system and results is WCFG. In WGCS, all grammar rules are in Chomsky Normal Form (CNF). The induction scheme of this model is shown in Figure 1 and the overall system architecture is shown in Figure 2.

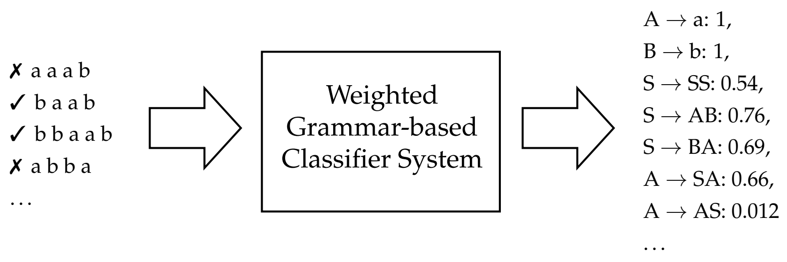

Figure 1. Induction in WGCS model.

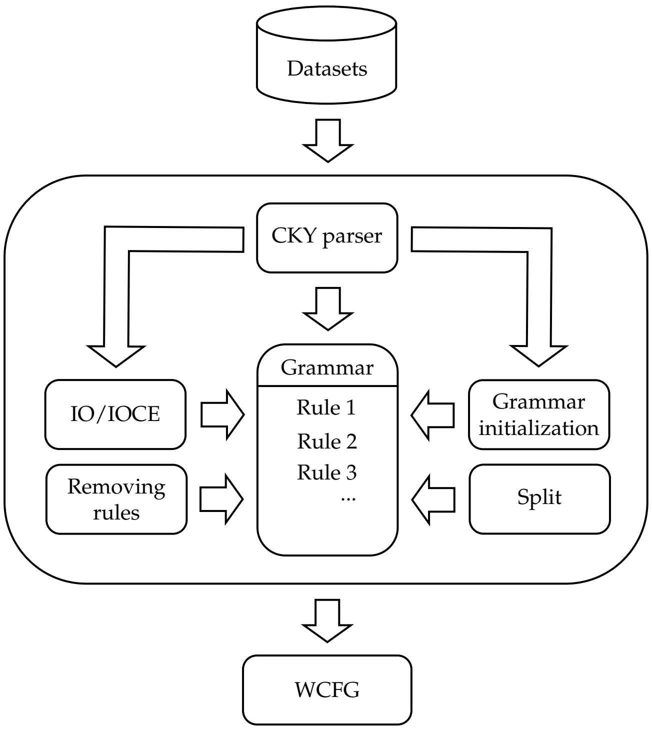

Figure 2. General architecture of the WGCS system.

2.1. Weighted Context-Free Grammar

A context-free grammar is a quadruple (N, T, S, R), where N—a finite set of nonterminals symbols disjoint from T, T—a finite set of terminals symbols, $S \in N$ the start symbol, and R—a finite set of rules of the form $X \to \alpha$, where $X \in N$ and $\alpha \in (N \cup T)^*$. CFG is in

CNF when each rule takes one of the two following forms: $X \to Y\ Z$ where $X, Y, Z \in N$ or $X \to t$ where $X \in N$ and $t \in T$.

A WCFG associates a positive number called the weight with each rule in R (assigning a weight of zero to a rule equates to excluding it from R). More formally, the WCFG is a 5-tuple (N, T, S, R, W), where (N, T, S, R) is a CFG and W is a finite set of weights of each rule resulting from a function $\phi(X \to \alpha) \to w$, where $X \to \alpha \in R$ and $w > 0$ is a positive weight.

2.2. Grammar Initialisation

Grammar in the WGCS system is initialised in two ways. The first way is to load a previously prepared grammar from a file. The second way is to generate it automatically in the application. Based on the training set, the symbols in this set are terminal symbols of the grammar, while their uppercase representations are nonterminal symbols. Then, to generate the terminal rules, each nonterminal symbol is assigned to each terminal symbol. On the other hand, nonterminal rules are generated by all possible combinations of all nonterminal symbols. According to [15], the number of all production rules of the CFG is $O(L^3)$, where L is the length of the input sentence. However, it should be noted that generating all productions is a one-time operation and does not affect the complexity of the method. In practice, the number of generated rules in the inference process is significantly less than the upper limit. Theoretically, the number of production rules in comparative methods (LS and ADIOS) is bounded by $O(L)$.

2.3. Stochastic CKY Parser

To verify whether a given sentence belongs to a specific grammar, special algorithms called parsers are used. One of the most common CFG parsers based on dynamic programming is the Cocke–Kasami–Younger (CKY) parser [16,17]. Its extension used to parse stochastic CFGs is Stochastic CKY, first described in [18]. Both classical and stochastic CKY algorithms assume grammar to be in CNF. The stochastic CKY algorithm is used in WGCS system and its operation is represented by Algorithm 1.

2.4. Split Algorithm

This method is based on [19,20], where grammar is induced incrementally. During each iteration of the algorithm, a new X_j nonterminal symbol is created from another X_i nonterminal symbol through a split operation. The symbol selected for the split operation is the symbol most often used in rule weight estimation (having the largest count) (see line 2 in Algorithm 2). This symbol is generally called the split symbol. Then, for all $X_i \to t$ terminal rules, we create new terminal rules, replacing X_i with X_j (see lines 3–5 in Algorithm 2). Next, new nonterminal rules are created in two ways:

1. For symbols X_i and X_j, create all possible rules (see line 6 in Algorithm 2). Since the rules are in CNF, their number is 8.
2. For all nonterminal rules with X_i in the form $X_a \to X_b X_c$, where X_a, X_b or X_c is X_i, create new untimely rules in the same form, replacing X_i with X_j. For multiple occurrences of X_i in a rule, create all combinations (see lines 7–20 in Algorithm 2).

Algorithm 1 Stochastic CKY

```
Load the sentence and grammar
$L \leftarrow$ sentence length
$|N| \leftarrow$ number of nonterminal symbols
Create an array CKY[$L$][$L$]
Create an array wCKY[$L$][$L$][$|N|$]

for $i \leftarrow 1$ to $L$ do
    for $w_i$ in $sentence$ do                                    ▷ $sentence = w_1 \dots w_L$
        for $A \rightarrow w_i$ in $TR$ do                         ▷ $TR = terminal\ rules$
            write $A$ in $CKY[i][1]$
            write $\phi(A \rightarrow w_i)$ in $wCKY[i][1][A]$     ▷ rule weight
        end for
    end for
end for

for $i \leftarrow 2$ to $L$ do
    for $j \leftarrow 1$ to $L - i + 1$ do
        for $k \leftarrow 2$ to $i - 1$ do
            for $A \rightarrow BC$ in $NR$ do                     ▷ $NR = non\text{-}terminal\ rules$
                if $wCKY[i][k][B] > 0$ and $wCKY[k][j][C] > 0$ then
                    write $A$ in $CKY[i][j]$
                    write $\phi(A \rightarrow BC) \times wCKY[i][k][B] \times wCKY[k][j][C]$ in $wCKY[i][j][A]$
                end if
            end for
        end for
    end for
end for
```

To illustrate the split method, suppose there is the set of rules: $R = \{Y \rightarrow BC, Y \rightarrow YC, B \rightarrow DY, S \rightarrow YY, Y \rightarrow a, B \rightarrow b, Y \rightarrow b\}$, the set of nonterminals $N = \{Y, B, C, D\}$, the set of terminals $T = \{a, b\}$, and the start symbol S. We select the symbol with the largest count (let it be Y) and create a new symbol—a split symbol—Z. According to the lines 3–5 of the Algorithm 2, new terminal rules are generated: $\{Z \rightarrow a, Z \rightarrow b\}$. The new nonterminal rules are generated as follows.

1. From Y and split symbol Z create: $Y \rightarrow YY$, $Y \rightarrow YZ$, $Y \rightarrow ZY$, $Y \rightarrow ZZ$, $Z \rightarrow ZZ$, $Y \rightarrow YZ$, $Z \rightarrow ZY$, $Z \rightarrow YY$ (line 6 of Algorithm 2).
2. From $\{Y \rightarrow BC, B \rightarrow DY\}$ create new following rules $\{Z \rightarrow BC, B \rightarrow DZ\}$ and from $\{Y \rightarrow YC, S \rightarrow YY\}$ rules $\{Z \rightarrow ZC, Z \rightarrow YC, Y \rightarrow ZC, S \rightarrow ZZ, S \rightarrow ZY, S \rightarrow YZ\}$ (lines 7–21 of Algorithm 2).

The result is: $R = \{Y \rightarrow YY, Y \rightarrow YZ, Y \rightarrow YZ, Y \rightarrow ZY, Y \rightarrow ZZ, Z \rightarrow ZZ, Z \rightarrow ZY, Z \rightarrow YY, Z \rightarrow ZC, Z \rightarrow YC, Y \rightarrow ZC, S \rightarrow ZZ, S \rightarrow ZY, S \rightarrow YZ, Y \rightarrow BC, Y \rightarrow YC, B \rightarrow DY, S \rightarrow YY, Y \rightarrow a, B \rightarrow b, Y \rightarrow b, Z \rightarrow a, Z \rightarrow b\}$, $N = \{Z, Y, B, C, D\}$, $T = \{a, b\}$, $S = S$.

Algorithm 2 Split algorithm

```
1:  Select a nonterminal symbol $X_i$ with the largest count    ▷ Split symbol
2:  Create new nonterminal symbol $X_j$
3:  for $X_i \rightarrow t$ in $TR$ do    ▷ $TR = terminal\ rules$
4:      Create new terminal rule $X_j \rightarrow t$
5:  end for
6:  Create all possible nonterminal rules from two nonterminal symbols $X_i$ and $X_j$:
    $X_i \rightarrow X_iX_i$, $X_i \rightarrow X_iX_j$, $X_i \rightarrow X_jX_i$, $X_i \rightarrow X_jX_j$, $X_j \rightarrow X_iX_i$, $X_j \rightarrow X_iX_j$, $X_j \rightarrow X_jX_i$,
    $X_j \rightarrow X_jX_j$
7:  for $X_a \rightarrow X_bX_c$ in $NR$ do    ▷ $NR = non\text{-}terminal\ rules$
8:      if $X_a == X_i \wedge X_b == X_i$ then
9:          Create new nonterminal rules: $X_j \rightarrow X_jX_c, X_j \rightarrow X_iX_c, X_i \rightarrow X_jX_c$
10:     else if $X_a == X_i \wedge X_c == X_i$ then
11:         Create new nonterminal rules: $X_j \rightarrow X_bX_j, X_j \rightarrow X_bX_i, X_i \rightarrow X_bX_i$
12:     else if $X_b == X_i \wedge X_c == X_i$ then
13:         Create new nonterminal rules: $X_a \rightarrow X_jX_j, X_a \rightarrow X_jX_i, X_a \rightarrow X_iX_j$
14:     else if $X_a == X_i$ then
15:         Create new nonterminal rule $X_j \rightarrow X_bX_c$
16:     else if $X_b == X_i$ then
17:         Create new nonterminal rule $X_a \rightarrow X_jX_c$
18:     else if $X_c == X_i$ then
19:         Create new nonterminal rule $X_a \rightarrow X_bX_j$
20:     end if
21: end for
```

2.5. Rule Weight Estimation

After establishing he grammar structure, we can focus on fine-tuning the weights of the rules. The most common algorithm used for this purpose is inside–outside. It is a special case of the Expectation-Maximization algorithm designed to estimate the parameters of a stochastic context-free grammar, originally the probabilities of the rules, and in our case, the rule weights. Two algorithms will be described in this subsection—the original inside–outside algorithm and its extended version using negative sentences when estimating rule weights used in the WGCS system.

2.5.1. Inside–Outside

Baker introduced the inside–outside algorithm [21]. Its computational complexity is $O(L^3|N|^3)$, where L is the sentence length and $|N|$ is the number of nonterminal symbols in the grammar [22].

The inside–outside algorithm starts the estimation process from the initial probabilities/weights of the rules (usually assigned randomly). At each iterative step, it updates the probability/weight of the rule based on the frequency of the rule in the training set. To better understand the algorithm, let us introduce the basic nomenclature.

- **Probability/weight of the rule**:
  - for nonterminal rules: $\phi(X \rightarrow YZ)$
  - for terminal rules: $\phi(X \rightarrow x)$

- **Probability/weight of deriving a sentence from grammar**:

$$P(W) = P(S \Rightarrow w_1 w_2 \dots w_n) \tag{1}$$

where $\Rightarrow$ stands for sentence derivation, W stands for sentence output, and $w_1 w_2 \dots w_n$ are the individual words of the sentence W.

- **The inside probability** is the probability of deriving from a given symbol the nonterminal sequence of words $w_i \dots w_j$ from the sentence $W = w_1 \dots w_n$:

$$\beta_{ij}(X) = P(X \Rightarrow w_i \dots w_j) \tag{2}$$

where X is any nonterminal grammar symbol.
Figure 3 shows the graphical interpretation of the inside probability for the nonterminal Y symbol, $\beta_{ij}(Y)$.

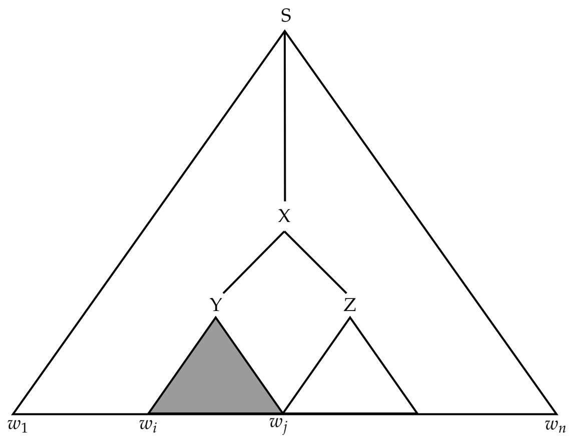

Figure 3. Graphical representation of the inside probability.

- **The outside probability** is the probability of deriving from the starting symbol the string $w_1 \dots w_{i-1} X w_{j+1} \dots w_n$ from the sentence $W = w_1 \dots w_n$:

$$\alpha_{ij}(X) = P(S \Rightarrow w_1 \dots w_{i-1} X w_{j+1} \dots w_n) \tag{3}$$

Figure 4 shows the graphical representation of the outside probability for the nonterminal symbol Y, $\alpha_{ij}(Y)$.

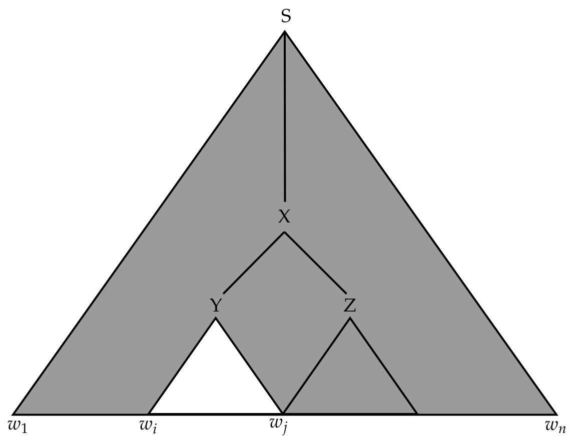

Figure 4. Graphical representation of the outside probability.

- **Estimated number of uses of the rule** determines how often the rule occurs for a single sentence:

- for terminal rules:

$$c_\phi(X \to x, W) = \frac{\phi(X \to x)}{P(W)} \sum_{i \le 1} \beta_{ii}(X) \tag{4}$$

- for nonterminal rules:

$$c_\phi(X \to YZ, W) = \frac{\phi(X \to YZ)}{P(W)} \sum_{1 \le i \le j \le k \le n} \alpha_{ik}(X)\beta_{ij}(Y)\beta_{j+1,k}(Z) \tag{5}$$

- **The total estimated number of uses of the rule** determines how often the rule occurs in all sentences in the training set:

$$count(X \to \alpha) = \sum_{i=1}^{n} c_\phi(X \to \alpha, W_i) \tag{6}$$

  where $X \to \alpha$ stands for a terminal or nonterminal rule and W_i for successive sentences in the training set.
- **The new weight/probability of a rule** is calculated as the ratio of the total estimated uses of a given rule to the sum of the estimated total uses of a rule with the same left-hand symbol:

$$\phi'(X \to \alpha) = \frac{count(X \to \alpha)}{\sum_\gamma count(X \to \gamma)} \tag{7}$$

  where $X \to \gamma$ stands for a rule with the same left-hand side symbol as the rule in the numerator and any right-hand side form (*gamma*) of a rule in a CNF, either a terminal symbol or two nonterminal symbols.

2.5.2. Inside–Outside Contrastive Estimation

The inside–outside contrastive estimation (IOCE) algorithm is an extended version of the inside–outside algorithm to include the use of negative sentences in the estimation of rule weights. This approach is inspired by works using the classic contrastive estimation method [23,24]. However, it differs significantly from the solutions proposed in those works.

In IOCE, we introduce the so-called negative estimation factor:

$$\psi(X \longrightarrow \alpha) = \frac{count(X \longrightarrow \alpha)}{count(X \longrightarrow \alpha) + \theta \times count_{negative}(X \longrightarrow \alpha)} \tag{8}$$

where $count_{negative}(X \longrightarrow \alpha)$ determines the total estimated number of uses of the rule in all negative sentences in the training set and $\theta = \frac{number\_of\_positive\_sentences}{number\_of\_negative\_sentences}$ is introduced to mitigate imbalanced datasets bias.

Using this coefficient, we calculate the new weight of the rule:

$$\varphi'(X \longrightarrow \alpha) = \frac{count(X \longrightarrow \alpha)}{\sum_\beta count(X \longrightarrow \beta)} \cdot \psi(X \longrightarrow \alpha) \tag{9}$$

The general idea of the negative estimation coefficient is that, if the rule often appears in the set of negative sentences, the coefficient is smaller. By multiplying the coefficient by the current weight of the rule, we reduce its weight. When the rule does not appear even once in the set of negative sentences, the coefficient is equal to 1 and the weight of the rule does not change.

2.6. Removing Rules with Low Weight

In the WGCS system with a split algorithm, the grammar size increases with each iteration, which significantly affects the computational complexity of IO/IOCE algorithms. Therefore, to prevent slowing down of the system and maintain good quality sentence classification, the WGCS system has implemented a mechanism that cleans the grammar from rules with low weights. If a rule gains a weight below the threshold, the rule is removed from the system. There are two thresholds for deleting rules, one for nonterminal rules 0.001 and another for terminal rules 0.000001. These values have been determined experimentally.

2.7. Experimental Protocol and Complexity Analysis

WCFG is learned using WGCS according to the experimental protocol described in Algorithm 3. The run-time complexity for the worst-case scenario of the given algorithm can be evaluated as follows. Say that k is the number of iterations of the WGCS algorithm, $|G|$—the size of the grammar, z—the number of sentences in the training set, n—the number of IOCE iterations, $|N|$—the number of nonterminal symbols, L—the length of the sentence, and y—the number of sentences in the validation set.

Algorithm 3 Experimental protocol

```
1: Load the initial grammar and datasets
2: for $i \leftarrow 1$ to $iterations$ do                ▷ iterations = 20
3:     Run the split algorithm
4:     Run IOCE on the training set                       ▷ 200
5:     Remove rules with low weights
6:     Test grammar with CKY on the training set
7: end for
8: Test best grammar with CKY on the validation set
```

In the algorithm above, for a worst-case evaluation it should be assumed that run-time of the split algorithm (line 3) is bounded by $|G|$, IOCE algorithm (line 4) is bounded by $z \cdot n \cdot |N|^3 L^3$, removing rules (line 5) by $|G|$, testing grammar (line 6) by $z \cdot L^3|G|$, and testing the best grammar (step 8) by $y \cdot L^3|G|$. Thus the total amount of time to run lines 1–8 is:

$$k \cdot (|G| + z \cdot n \cdot |N|^3 L^3 + |G| + z \cdot L^3|G|) + y \cdot L^3|G| \tag{10}$$

Note that $|G| = k \cdot |T| + k^3$ and $|N| = k^3$, and the total amount of time can be calculated as follows

$$k \cdot (k \cdot |T| + k^3 + z \cdot n \cdot k^3 L^3 + k \cdot |T| + k^3 + z \cdot L^3(k \cdot |T| + k^3)) + y \cdot L^3(k \cdot |T| + k^3) \tag{11}$$

which can be factored as:

$$k^2 \cdot |T| + k^4 + z \cdot n \cdot k^4 L^3 + k^2 \cdot |T| + k^4 + z \cdot L^3(k^2 \cdot |T| + k^4) + y \cdot L^3(k \cdot |T| + k^3) \tag{12}$$

Therefore, the total running time for this algorithm is estimated as:

$$O(z \cdot n \cdot |T| \cdot k^4 L^3 + y \cdot |T| \cdot k^3 L^3) \tag{13}$$

Note that the maximum number of terminals $|T|$ can be replaced by $(y + z)L$

$$O(z \cdot n \cdot (y + z) \cdot L \cdot k^4 L^3 + y \cdot (y + z) \cdot L \cdot k^3 L^3) \tag{14}$$

which reduces to

$$O((y+z)\cdot k^3L^4(z\cdot n\cdot k+y)) \tag{15}$$

As we can see the complexity of the proposed method is polynomially bounded with respect to the input size.

3. Benchmarks

3.1. Datasets

For our experiments, 28 datasets have been prepared. Most of them were generated based on the random context-free grammars G_i obtained from the CFG/TestSet Generator [25]. The target finite samples, that had about 200 words each were constructed as follows. Let $Z_i = \bigcup_{k=1}^{K_i} L_i \cap \Sigma^k$. The total estimated number of uses for the rule, where K_i is an integer from 10 to 20 and $L_i = L(G_i)$. 100 words, chosen randomly from the set Z_i, along with optimal examples given by the generator constituted examples. Let $z \in Z_i$ and $y \in \Sigma^*$ be words that differ by a few letters—as a consequence of a swap, insertion, or deletion. 100 words $y \in Y_i$, $y \notin L_i$, $1 \leq |y| \leq K_i$, generated randomly in this way, constituted counterexamples.

Five languages were generated based on grammars constructed by hand from the following description:

- L_6 : balanced parentheses
- L_8 : $\{w\colon w$ is a palindrome and $w \in \{a,b\}\{a,b\}^+\}$
- L_9 : $\{w\colon w \in \{a,b\}^+$ and $\sharp_a(w) = \sharp_b(w)\}$
- L_{10} : $\{w\colon w \in \{a,b\}^+$ and $2\sharp_a(w) = \sharp_b(w)\}$
- L_{11} : the language of Łukasiewicz ($S \to aSS;\ S \to b$)

Languages L_6, L_8, L_9, and L_{10} were considered by Nakamura and Matsumoto [26], and L_{11} was considered by Eyraud et al. [27]. Table 1 shows our settings in this respect.

Table 1. Datasets metrics.

Dataset	Size	Positive Sentences	Negative Sentences	Max. Length of Sentence	Min. Length of Sentence	Number of Terminals
1	213	113	100	18	3	4
2	220	120	100	18	2	2
3	204	104	100	14	4	2
4	240	140	100	20	4	5
5	208	108	100	20	8	4
6	200	100	100	20	12	2
7	198	98	100	20	4	4
8	200	100	100	14	3	2
9	200	100	100	16	6	2
10	200	100	100	20	11	2
11	200	100	100	20	1	2
12	204	104	100	20	4	6
13	205	105	100	20	3	4
14	200	100	100	20	3	3
15	200	100	100	20	3	5
16	216	116	100	20	2	7
17	197	97	100	20	3	3
18	206	106	100	20	5	5
19	240	140	100	12	2	2
20	209	109	100	20	6	4
21	213	113	100	20	5	6
22	205	105	100	20	7	4
23	209	109	100	20	5	5
24	199	99	100	20	3	3
25	207	107	100	16	3	6
26	200	100	100	20	2	2
27	190	90	100	16	2	5
28	224	124	100	18	5	6

3.2. *Brief Description of Other Approaches*

In [13], Wieczorek described a local search (LS) approach. In this study, we will use a simple example to present this method using a simple example. Assume that the set $\{ab, abab\}$ constitutes examples while the set $\{a, b, ba, aba\}$ constitutes counterexamples. The first step is to construct a grammar that generates all the examples. A special algorithm has been devised for this purpose. It could produce the following grammar: $S \rightarrow AB$, $A \rightarrow a$, $B \rightarrow CD$, $C \rightarrow b$, $D \rightarrow AC \mid E$, and $E \rightarrow \epsilon$. Further, in a loop, two variables are merged as long as the grammar can be shortened. In the examples, B and C (into B), S and D, and S and E, we get the grammar: $S \rightarrow AB \mid \epsilon$, $A \rightarrow a$, $B \rightarrow b \mid BS$. Finally, unnecessary variables and rules are removed from the resultant grammar (in the example the rule $S \rightarrow \epsilon$). Every step is controlled by means of counterexamples to obtain valid grammar.

The Automatic DIstillation Of Structure (ADIOS) model that uses only examples builds syntactic representations of a sample of language from unlabelled data [28]. It consists of two elements: (1) a Representational Data Structure (RDS) graph and (2) a Pattern Acquisition (PA) algorithm that constructs the RDS in an unsupervised manner. The goal of the PA algorithm is to detect patterns, i.e., repetitive sequences of "significant" strings occurring in the examples. Here, the PA algorithm is related to prior work on alignment-based learning and regular expression extraction from strings. However, the authors of ADIOS stress claim, that their algorithm requires no prejudging of either the scope of the primitives or their classification. In the initial phase of the PA algorithm, the examples are segmented down to the smallest possible morphological constituents. In the second phase, the PA algorithm repeatedly scans the RDS graph for common patterns, which are then used to modify the graph. ADIOS algorithm has been tested on a variety of linguistic and bioinformatics data with promising results.

The code of WGCS and LS along with the benchmarks are available at [29]. The code of ADIOS is available on request from the authors of this method.

4. Results

Our experiments were performed on Intel Core i7-7567U CPU, 3.5 GHz processor, with 32 GB RAM under Windows 10 operating system. Three methods, i.e., our proposal WGCS and two references methods: local search (LS) and ADIOS, were used to infer grammars for 28 benchmark datasets. A five-fold crossvalidation was performed on each set and the results were averaged. To evaluate the quality classification of the compared methods, we use the classification results stored in a confusion matrix. Four scores were defined as tp, fp, fn, and tn, representing the numbers of true positives (correctly recognised positive sentences), false positives (negatives recognised as positives), false negatives (positives recognised as negatives), and true negatives (correctly recognised negatives), respectively. Based on the values stored in the confusion matrix, we calculate the widely used Precision, Recall (Sensitivity), and combined metric F1-score. Precision is defined as $P = tp/(tp + fp)$, Recall (Sensitivity) as $R = tp/(tp + fn)$, and F1 as the harmonic mean of Precision and Sensitivity $F1 = 2 \cdot (P \cdot R/(P + R))$. Table 2 shows these results with respect to Precision (Pr), Recall (Rc), and F1 score. This table additionally shows the average production number and the average computation time obtained for each tested grammar by all methods. The average number of productions and the average computation time were calculated over five folds of the crossvalidation method used.

To find out whether the observed differences are statistically significant, we follow the Wilcoxon-signed-rank test [30] for WGCS vs. LS and WGCS vs. ADIOS. In this test, the null hypothesis (H_0) states that the difference between the pairs follows a symmetric distribution around zero. The alternative hypothesis (H_1), on the other hand, states that the difference between the pairs does not follow a symmetric distribution around zero (i.e., the difference is not a coincidence). As can we see in Table 3, p-values are small enough (all below 0.025) to reject H_0. Therefore, we can conclude that WGCS performs better than the two competing methods on prepared benchmarks, although it is a slower method than ADIOS.

Table 2. Average grammar size (∣G∣), Precision (Pr), Recall (Rc), F1, and average time ([[hh:]mm:]ss) for WGCS, LS and ADIOS.

Set	WGCS					LS					ADIOS				
	∣G∣	Pr	Rc	F1	Time	∣G∣	Pr	Rc	F1	Time	∣G∣	Pr	Rc	F1	Time
1	25	0.99	1.00	0.99	19:18	16.8	1.00	0.99	0.99	26:59:07	28.2	0.90	0.98	0.94	1.2
2	47	0.99	0.98	0.99	14:54	75.0	0.98	0.88	0.93	97:54:37	33.8	0.63	1.00	0.77	2.0
3	16	1.00	1.00	1.00	1:43	34.8	0.99	0.90	0.94	56:44:14	34.4	0.51	1.00	0.68	2.4
4	40.6	0.98	0.99	0.97	16:12	19.4	1.00	1.00	1.00	7:22:35	26.4	0.60	1.00	0.75	3.0
5	38	0.98	0.98	0.98	1:41:53	10.2	1.00	1.00	1.00	19:06:57	30.0	0.52	1.00	0.68	4.5
6	11.2	1.00	1.00	1.00	5:43	132.0	1.00	0.48	0.61	4:37:01	25.2	0.5	1.00	0.67	4.3
7	19.2	1.00	1.00	1.00	48:31	28.2	1.00	0.94	0.97	2:50:37	26.4	0.49	1.00	0.66	4.7
8	56.6	0.94	0.95	0.94	2:21:43	92.9	0.79	0.51	0.61	89:22:16	24.2	0.50	1.00	0.67	4.0
9	41.2	0.76	0.90	0.81	1:51:15	192.4	0.20	0.02	0.04	2:24:38	20.6	0.50	1.00	0.67	6.9
10	42.4	0.85	0.93	0.88	4:13:26	192.2	0.20	0.01	0.02	2:21:31	31.0	0.50	1.00	0.67	5.9
11	7.8	1.00	0.99	0.99	1:05:39	67.4	0.99	0.78	0.86	16:53:53	29.6	0.50	1.00	0.67	6.0
12	21.4	0.97	0.99	0.98	1:23:47	16.0	1.00	0.99	0.99	1:59:20	24.2	0.56	1.00	0.72	6.0
13	40	0.97	0.99	0.98	1:05:18	9.8	1.00	0.99	0.99	15:26:20	27.8	0.51	1.00	0.68	7.8
14	42.2	0.99	0.98	0.98	1:27:58	84.0	0.96	0.71	0.80	49:18:12	27.8	0.50	1.00	0.67	8.7
15	57	0.96	0.91	0.93	1:22:26	26.0	0.99	0.92	0.95	31:39:10	23.2	0.50	1.00	0.67	7.7
16	57.2	0.98	0.93	0.96	47:29	12.8	1.00	1.00	1.00	4:18:35	32.2	0.56	0.98	0.71	10.5
17	40.2	0.97	0.99	0.98	52:33	93.5	0.98	0.82	0.88	86:43:13	20.6	0.49	1.00	0.66	8.4
18	27.6	1.00	1.00	1.00	5:53	26.2	1.00	0.91	0.95	2:46:58	21.2	0.51	1.00	0.68	9.3
19	65.4	0.96	0.89	0.92	17:03	82.9	0.84	0.61	0.70	88:28:50	23.4	0.58	0.00	0.73	8.5
20	64	0.96	0.90	0.93	1:56:21	12.4	1.00	0.99	0.99	75:31:18	20.8	0.52	1.00	0.69	10.2
21	46.8	0.99	1.00	0.99	55:27	15.2	1.00	1.00	1.00	62:32:23	25.0	0.60	1.00	0.74	8.5
22	40.2	0.98	1.00	0.99	1:08:24	14.0	1.00	0.96	0.98	17:09:24	24.0	0.51	1.00	0.68	10.7
23	23.4	0.97	0.99	0.98	20:59	14.8	1.00	0.99	0.99	46:38:02	25.0	0.52	1.00	0.69	10.5
24	50.6	0.88	0.95	0.91	1:17:20	75.8	0.98	0.71	0.82	28:55:28	28.0	0.50	1.00	0.66	11.5
25	54.6	0.98	0.88	0.92	16:08	17.0	1.00	0.99	0.99	38:25:13	24.2	0.52	1.00	0.68	12.7
26	48	0.87	0.97	0.91	2:15:20	122.0	0.99	0.53	0.68	84:07:51	28.4	0.50	1.00	0.67	16.0
27	29.8	0.99	0.98	0.98	4:43	18.2	1.00	0.96	0.98	18:43:47	21.0	0.47	1.00	0.64	12.3
28	30.4	0.96	0.97	0.98	18:07	25.8	1.00	1.00	1.00	18:21:58	23.4	0.55	1.00	0.71	11.2
Avg	38.7	0.96	0.97	0.96	1:01:59	54.6	0.92	0.81	0.85	35:37:59	26.1	0.54	0.96	0.70	7.7

Table 3. Obtained p values for F1 from Wilcoxon signed-rank test.

WGCS vs. LS	WGCS vs. ADIOS	LS vs. ADIOS
1.87×10^{-2}	3.74×10^{-6}	1.81×10^{-3}

5. Conclusions

We have formalized and verified the split method in weighted context-free grammar induction. The new approach to weighted CFG learning has been applied in the frame of the Weighted Grammar-based Classifier System. Additionally, the inside–outside contrastive estimation algorithm was improved by correcting unbalanced data bias. The experiments conducted over 28 context-free languages showed that WGCS with splitting outperforms the state-of-the-art methods in terms of F1 scores.

Further work is ongoing to investigate the use of the combined split-merge method in discovering WCFG. It should be noted that the split mechanism leads to an overlinear increase in the number of generated productions. It is also worth noting that grammar splitting is focused on specialising grammar production, whereas grammar merging can generalise the model by merging some nonterminals. Grammar merging seems to be a promising approach to pruning unwanted structures [31].

Author Contributions: Conceptualisation, O.U. and M.G.; methodology, O.U. software, M.G. and W.W.; validation, M.G. and O.U.; formal analysis, O.U.; investigation, M.G. and O.U.; resources, M.G. and W.W.; writing—original draft preparation, M.G. and W.W. and O.U.; writing—review and editing, O.U.; supervision, O.U.; project administration, O.U.; funding acquisition, O.U. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by National Science Center (Poland), grant number 2016/21/B/ST6/02158.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available in [29].

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

IO	Inside-Outside algorithm
CFG	Context-Free Grammar
CNF	Chomsky Normal Form
CKY	Cocke–Kasami–Younger parser
WCFG	Weighted Context-Free Grammar
WGCS	Weighted Grammar-based Classifier System
IOCE	Inside-Outside Contrastive Estimation algorithm

References

1. Flasiński, M. *Introduction to Artificial Intelligence*; Springer: Berlin/Heidelberg, Germany, 2016.
2. de la Higuera, C. *Grammatical Inference: Learning Automata and Grammars*; Cambridge University Press: Cambridge, UK, 2010; doi:10.1017/CBO9781139194655. [CrossRef]
3. Gold, E.M. Language identification in the limit. *Inf. Control.* **1967**, *10*, 447–474. [CrossRef]
4. Horning, J.J. *A Study of Grammatical Inference*; Technical Report; Stanford University California Department of Computer Science: Stanford, CA, USA, 1969.
5. Unold, O.; Gabor, M.; Wieczorek, W. Unsupervised Statistical Learning of Context-free Grammar. In *Proceedings of the 12th International Conference on Agents and Artificial Intelligence—Volume 1: NLPinAI; INSTICC*; SciTePress: Setúbal, Portugal, 2020; pp. 431–438.
6. Unold, O.; Gabor, M.; Dyrka, W. Unsupervised Grammar Induction for Revealing the Internal Structure of Protein Sequence Motifs. In Proceedings of the International Conference on Artificial Intelligence in Medicine, Minneapolis, MN, USA, 25–28 August 2020; Springer: Berlin/Heidelberg, Germany, 2020; pp. 299–309.
7. Unold, O. Context-free grammar induction with grammar-based classifier system. *Arch. Control. Sci.* **2005**, *15*, 681–690.
8. Unold, O. Fuzzy grammar-based prediction of amyloidogenic regions. In Proceedings of the International Conference on Grammatical Inference, College Park, MD, USA, 5–8 September 2012; pp. 210–219.
9. Unold, O.; Gabor, M. How implicit negative evidence improve weighted context-free grammar induction. In Proceedings of the International Conference on Artificial Intelligence and Soft Computing, Zakopane, Poland, 16–20 June 2019; Springer: Berlin/Heidelberg, Germany, 2019; pp. 595–606.
10. Van Zaanen, M. ABL: Alignment-based learning. In Proceedings of the 18th Conference on Computational Linguistics, Saarbrucken, Germany, 31 July–4 August 2000; Volume 2, pp. 961–967.
11. Adriaans, P.; Vervoort, M. The EMILE 4.1 grammar induction toolbox. In Proceedings of the International Colloquium on Grammatical Inference, Amsterdam, The Netherlands, 23–25 September 2002; Springer: Berlin/Heidelberg, Germany, 2002; pp. 293–295.
12. Solan, Z.; Horn, D.; Ruppin, E.; Edelman, S. Unsupervised learning of natural languages. *Proc. Natl. Acad. Sci. USA* **2005**, *102*, 11629–11634. [CrossRef]
13. Wieczorek, W. A Local Search Algorithm for Grammatical Inference. In *Grammatical Inference: Theoretical Results and Applications, Proceedings of the 10th International Colloquium (ICGI 2010), Valencia, Spain, 13–16 September 2010*; Lecture Notes in Computer Science; Jose, M., Sempere, P.G., Eds.; Springer: Berlin/Heidelberg, Germany, 2010; Volume 6339, pp. 217–229.
14. Urbanowicz, R.J.; Moore, J.H. Learning classifier systems: A complete introduction, review, and roadmap. *J. Artif. Evol. Appl.* **2009**, *2009*, 1. [CrossRef]
15. Sakakibara, Y. Learning context-free grammars using tabular representations. *Pattern Recognit.* **2005**, *38*, 1372–1383. [CrossRef]
16. Kasami, T. *An Efficient Recognition and Syntax-Analysis Algorithm for Context-Free Languages*; Coordinated Science Laboratory Report No. R-257; University of Illinois at Urbana-Champaign: Champaign, IL, USA, 1966.
17. Younger, D.H. Recognition and parsing of context-free languages in time n3. *Inf. Control* **1967**, *10*, 189–208. [CrossRef]
18. Ney, H. Dynamic programming parsing for context-free grammars in continuous speech recognition. *IEEE Trans. Signal Process.* **1991**, *39*, 336–340. [CrossRef]
19. Hogenhout, W.R.; Matsumoto, Y. A fast method for statistical grammar induction. *Nat. Lang. Eng.* **1998**, *4*, 191–209. [CrossRef]
20. Kurihara, K.; Sato, T. Variational Bayesian grammar induction for natural language. In Proceedings of the International Colloquium on Grammatical Inference, Tokyo, Japan, 20–22 September 2006; Springer: Berlin/Heidelberg, Germany, 2006; pp. 84–96.
21. Baker, J.K. Trainable grammars for speech recognition. *J. Acoust. Soc. Am.* **1979**, *65*, S132. [CrossRef]
22. Lari, K.; Young, S.J. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Comput. Speech Lang.* **1990**, *4*, 35–56. [CrossRef]

23. Smith, N.A.; Eisner, J. Contrastive estimation: Training log-linear models on unlabeled data. In Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics, Ann Arbor, MI, USA, 25–30 June 2005; pp. 354–362.
24. Smith, N.A.; Eisner, J. Guiding unsupervised grammar induction using contrastive estimation. In Proceedings of the IJCAI Workshop on Grammatical Inference Applications, Edinburgh, UK, 31 July 2005; pp. 73–82.
25. Unold, O.; Kaczmarek, A.; Culer, Ł. Iterative method of generating artificial context-free grammars. *arXiv* **2019**, arXiv:1911.05801.
26. Nakamura, K.; Matsumoto, M. Incremental learning of context free grammars based on bottom-up parsing and search. *Pattern Recognit.* **2005**, *38*, 1384–1392. [CrossRef]
27. Eyraud, R.; de la Higuera, C.; Janodet, J.C. LARS: A learning algorithm for rewriting systems. *Mach. Learn.* **2007**, *66*, 7–31. [CrossRef]
28. Solan, Z.; Ruppin, E.; Horn, D.; Edelman, S. Automatic Acquisition and Efficient Representation of Syntactic Structures. In *Neural Information Processing Systems 15, Proceedings of the Neural Information Processing Systems (NIPS 2002), Vancouver, BC, Canada, 9–14 December 2002*; Becker, S., Thrun, S., Obermayer, K., Eds.; MIT Press: Cambridge, MA, USA, 2002; pp. 91–98.
29. Unold, O. jGCS. 2019. Available online: https://github.com/ounold/jGCS (accessed on 22 January 2021).
30. Rey, D.; Neuhäuser, M. Wilcoxon-Signed-Rank Test. In *International Encyclopedia of Statistical Science*; Lovric, M., Ed.; Springer: Berlin/Heidelberg, Germany, 2011; pp. 1658–1659. [CrossRef]
31. Zaanen, M.; Noord, N. Model merging versus model splitting context-free grammar induction. In Proceedings of the International Conference on Grammatical Inference, College Park, MD, USA, 5–8 September 2012; pp. 224–236.