April 2013 Technical Report: KUL/ESAT/PSI/1301

Early Parsing for 2D Stochastic Context Free Grammars

Andelo Martinovic and Luc Van Gool

PSI

Kasteelpark Arenberg 10 – box 2441
B-3001 Heverlee, Belgium

KU LEUVEN

Telephone: +32-(0)16-32.17.13 Fax: +32-(0)16-32.17.23 http://www.esat.kuleuven.be/psi

Earley Parsing for 2D Stochastic Context Free Grammars Technical Report

Anđelo Martinović and Luc Van Gool

Abstract

In a companion paper, ***Bayesian Grammar Learning for Inverse Procedural Modeling,*** *we have presented a novel approach of learning a specific variant of procedural grammars from data, namely 2D Attributed Stochastic Context Free Grammars, or 2D-ASCFGs. One of the essential parts of the grammar learning algorithm described therein is the Earley parser for 2D-SCFGs. The purpose of this technical report is to provide the implementation details of this parser, illustrate the approach on simple examples, and discuss the improvements over the existing methods.*

1. Introduction

This technical report provides detailed instructions on the implementation and usage of an Earley-style parser for two-dimensional stochastic context free grammars (2D-SCFGs), used in our companion paper [6]. Earley's parser [4] is a well known top-down parsing algorithm for context free grammars (CFGs) that has a worst-case complexity of $O(n^3)$, where n is the size of the input. However, it was shown that the parser can perform substantially better for many well-known grammar classes. For example, it runs in quadratic time for unambiguous grammars, and in linear time for most LR(k) grammars. Another appealing property of Earley's algorithm is that it deals with any context-free rule format; it does not require grammar conversion to Chomsky Normal Form (CNF), as some alternatives do, such as CKY [10] chart parsing. Although every context-free grammar can be transformed into CNF, this comes with a price. Such conversion can lead to an undesirable increase of the grammar size [5].

Originally, Earley's parser was designed in the field of computational linguistics as a string parsing algorithm. The parser uses a dynamic programming approach that decides whether a given one-dimensional structure (i.e. a string) belongs to a given CFG. This original formulation was improved upon by various authors over time. The work of Stolcke [7] introduced an extension to stochastic context free grammars (SCFGs). Given the stochastic nature of these grammars, the upgraded parser output also includes the *probability* that a string is produced by the given grammar.

Some authors attempted to generalize the Earley parser by considering different input domains. In [3], relation grammars were proposed to model multi-dimensional structures. Wild [9] used multiple context free grammars (MCFGs) to model RNA interaction problems. Although similar in notation, their definition of a 2D-SCFG is very different from ours, since in MCFGs non-terminals produce tuples of words of given dimension. Costagliola and Chang [2] introduced positional grammars that explicitly model the spatial relations between symbols in grammar productions. This formalism was used for parsing arithmetical expressions. The approach closest to ours was presented by Tomita [8], who introduced a straightforward extension of the Earley parser to two-dimensional grid structures. However, some cases of input examples were not handled properly in this algorithm. In Section. 4 we show a failure example where the approach of [8] does not properly parse a 2D grid, in contrast to our proposed method. Furthermore, our methods generalizes to 2D stochastic CFGs, which are much less studied in literature. One of the few existing 2D SCFG parsing approaches was used for recognition of mathematical expressions [1]; however, this approach uses a 2D CKY parser, which requires the grammar to be in Chomsky Normal Form.

2. 2D-ASCFGs[1]

A two-dimensional attributed stochastic context-free grammar (2D-ASCFG) is defined as a tuple $G = (N, T, S, R, P, A)$, where N is a set of non-terminal symbols, T a set of terminal symbols, S the starting non-terminal symbol or axiom, R a set of production rules, $\{P(r), r \in R\}$ a set of rule probabilities and $\{A(r), r \in R\}$ a set of rule attributes.

[1] This section is an excerpt from [6].

Every symbol is associated with the corresponding shape, representing a rectangular region. Starting from the axiom, production rules subdivide the starting shape either in horizontal or vertical directions. We define the set R as a union of horizontal and vertical productions: $R = R_h \cup R_v$. These productions correspond to standard horizontal and vertical split operators in split grammars. A production is of the form $X \to \lambda$, where $X \in N$ is called the left-hand-side (LHS), and $\lambda \in (N \cup T)^+$ is called the right-hand-side (RHS) of the production.

For every production, $P(X \to \lambda)$ is defined as the probability that the rule is selected in the top-down derivation from the grammar. For the grammar to be well-formed, the productions with X as LHS must satisfy the condition $\sum_\lambda P(X \to \lambda) = 1$. Additionally, each grammar rule r is associated with a set of attributes $A(r) = \{\alpha_i\}$. The elements of a single attribute are the relative sizes of the RHS shapes in respect to their parent shape, in the splitting direction: $\alpha_i = \{s_1, ..., s_{|\lambda|}\}, \sum_i s_i = 1$. These relative sizes sum up to one because RHS shapes always fill the entire shape of their parent.

We denote by τ a parse tree from the grammar, rooted on the axiom, its interior nodes corresponding to non-terminal symbols, and its exterior nodes to terminal symbols. The parse tree is obtained by applying a sequence of rules on the axiom and non-terminal nodes. A derivation from the grammar consists of the parse tree and the selected attributes at each node: $\delta = (\tau, \alpha)$. The probability of a single derivation is the product of all rule probabilities selected at each node s of the parse tree: $P(\delta) = \prod_{s \in \delta} P(r_s)$. The set of terminal nodes of a parse tree defines a lattice over an area. A lattice, or a grid is a rectangular tessellation of 2D space, exactly filling the shape of the axiom. We define the likelihood of the grammar G generating a lattice l as $L(l|G) = \sum_{\delta \Rightarrow l} P(\delta)$, where we sum over the probabilities of all derivations that yield a particular lattice.

3. 2D Earley parser

In this section, we mostly follow the notation from [7] and [8], adapting it as necessary. For a set of input symbols organized in a grid I of width n and height m,

$$\begin{matrix} I_{11} & I_{21} & \dots & I_{n1} \\ I_{12} & I_{22} & \dots & I_{n2} \\ \vdots & & & \vdots \\ I_{1m} & I_{2m} & \dots & I_{nm} \end{matrix}$$

the parser keeps track of a set of *states* (alternatively called *items*) for each position in the input. Each of these states corresponds to a possible derivation of the grammar that is consistent with the input up to the given point in the grid. There can be multiple states at the same point in the grid, and they are denoted as a *state set.* All of these state sets together form the *Earley chart*. Unlike 1-dimensional Earley parsing, where the chart is a one-dimensional list of state sets, 2D Earley chart is a parse table of the form

$$\begin{matrix} S_{00} & S_{10} & \dots & S_{n0} \\ S_{01} & S_{11} & \dots & S_{n1} \\ \vdots & & & \vdots \\ S_{0m} & S_{1m} & \dots & S_{nm} \end{matrix}$$

Note that the state chart contains an additional row and column in comparison to the input. These correspond to the states when no symbols have been processed in the first input row or column. Each state is defined as a tuple (p, d, o, b):

- $p \in R_H \cup R_v$ is a production of the grammar.
- The *dot position* d is an index denoting the current position in the RHS of production $p(X \to \lambda\mu)$:

$$X \to \lambda.\mu$$

where $\lambda, \mu \in (N \cup T)^+$. This indicates that the RHS of the production was expanded up to the position indicated by the dot. The dot position $d = 0$ signifies that no RHS symbols have been analyzed so far. In contrast, the dot to the right of entire RHS corresponds to a fully expanded non-terminal X. Such a state is called *complete.*

- The *origin position* o is defined a pair of indices (i, j) corresponding to the position of the state in the Earley chart.

- The *bounding box* $b(x, y, X, Y)$ represents the admissible rectangular region of the input I for the current state. For all b, the bounding box must be non-empty : $x < X$, $y < Y$. Additionally, this region has to be contained within the input grid: $0 \leq x, X \leq n$, $0 \leq y, Y \leq m$. Finally, the origin position must always lie within the bounding box: $i < X$, $j < Y$.

We will use the following short-hand formulation to describe a state:

$$(i, j)\, X \to \lambda.\mu\, (x, y, X, Y)$$

To simplify the exposition of the algorithm, we convert the grammar into *Simple Normal Form* [7]. In this form, terminal symbols can appear only on the right-hand side of lexical productions, i.e. productions with a single terminal symbol on the RHS. All other productions can contain an arbitrary number of non-terminal symbols. This form can be easily achieved by simply "shadowing" every terminal symbol with an additional non-terminal symbol, e.g:

$$\begin{gathered} X \to aYc \\ \Downarrow \\ A \to a \\ C \to c \\ X \to AYC \end{gathered}$$

where $X, Y, A, C \in N$; $a, c \in T$. We have chosen, without loss of generality, the convention that lexical productions are contained in the set of horizontal productions. We also add a *starting production* to the grammar, designated as $\epsilon \to S$, where ϵ is an empty symbol and S is the axiom of the grammar.

3.1. The Parsing Algorithm

Now we have all the prerequisites to describe the 2D Earley parsing algorithm. At the beginning, an *initial state* is created and subsequently inserted into the $(0, 0)$ cell of the Earley chart*:*

$$(0, 0)\, \epsilon \to .S\, (0, 0, n, m)$$

Afterwards, the parser performs three operations: *prediction, scanning* and *completion*.

Prediction

Given an incomplete state (the dot is not in the final position), the role of *prediction* step is to enumerate all potential expansions of the non-terminal to the right of the dot. For each state

$$s: \quad (i, j)\, A \to \lambda.B\mu\, (x, y, X, Y)$$

the algorithm finds all possible expansions $B \to \nu$ of non-terminal B in the grammar and creates new states to be added to the chart, of the form:

$$s_{new}: \quad (i, j)\, B \to .\nu\, (i, j, X, Y)$$

Scanning

In the scanning step, we read the input symbol at the current position in the grid. All states that are in accordance with the input symbol are discovered. Note that, as the grammar is in SNF, the only states considered are the ones containing a lexical production. For each state

$$s: \quad (i, j)\, A \to .a\, (x, y, X, Y)$$

where $a \in T$ matches the input symbol $I_{i+1,j+1}$, add a new state by moving the dot over the scanned symbol:

$$s_{new}: \quad (i + 1, j)\, A \to a.\, (i, j, i + 1, j + 1)$$

The produced states have the dot on the far right of the RHS, and are thus complete.

Completion

In the previous step, a complete state was produced from a predicted state by scanning the input. The predecessors of the predicted state now need to be updated to take into account the scanned symbol. For each complete state

$$s: \quad (i,j)\, B \rightarrow \nu .\, (x,y,X,Y)$$

find candidate states s' in Earley chart cell (x,y), which have B as the non-terminal after the dot:

$$s': \quad (x,y)\, A \rightarrow \lambda . B \mu\, (x',y',X',Y')$$

The necessary condition for a candidate state to be considered is that its bounding box encompasses the bounding box of the complete state:

$$x' \leq x \wedge y' < y \wedge X' \geq X \wedge Y' \geq Y$$

Productions $p: \ B \rightarrow \nu$ and $p': \ A \rightarrow \lambda B \mu$ belong either to the set of horizontal or vertical productions, thus there are four different combinations:

1. $p \in \mathbf{R}_h, p' \in \mathbf{R}_h$

   If $Y' = Y \vee \lambda = \emptyset$, add state $s_{new} = (i,j)\, A \rightarrow \lambda B . \mu\, (x',y',X'',Y)$, where $X'' = \begin{cases} X, & if\ \mu = \emptyset \\ X', & otherwise \end{cases}$

2. $p \in \mathbf{R}_v, p' \in \mathbf{R}_h$

   If $Y' = Y \vee \lambda = \emptyset$, add state $s_{new} = (X,y)\, A \rightarrow \lambda B . \mu\, (x',y',X'',Y)$, where $X'' = \begin{cases} X, & if\ \mu = \emptyset \\ X', & otherwise \end{cases}$

3. $p \in \mathbf{R}_h, p' \in \mathbf{R}_v$

   If $X' = X \vee \lambda = \emptyset$, add state $s_{new} = (x,Y)\, A \rightarrow \lambda B . \mu\, (x',y',X,Y'')$, where $Y'' = \begin{cases} Y, & if\ \mu = \emptyset \\ Y', & otherwise \end{cases}$

4. $p \in \mathbf{R}_v, p' \in \mathbf{R}_v$

   If $X' = X \vee \lambda = \emptyset$, add state $s_{new} = (i,j)\, A \rightarrow \lambda B . \mu\, (x',y',X,Y'')$, where $Y'' = \begin{cases} Y, & if\ \mu = \emptyset \\ Y', & otherwise \end{cases}$

These conditions simply state that the bounding box edges of states s and s' must align vertically ($Y' = Y$), or horizontally ($X' = X$), unless s' has not been expanded. The definitions of X'' and Y'' are necessary if the newly created state s_{new} is complete ($\mu = \emptyset$). In that case, the bounding box of state s' must be clipped to the value given by s (see Figure 1). Additionally, the completion step will have to be performed on s_{new} as well.

A special case during completion occurs when symbol B is identical to the grammar axiom S, i.e. the candidate production is the starting production:

$$s: \quad (i,j)\, \epsilon \rightarrow .S\, (x',y',X',Y')$$

In that case, we add a state

$$s_{new}: \quad (i,j)\, \epsilon \rightarrow S .\, (x,y,X,Y)$$

If such a state is produced during parsing, one valid parse of the input grid was found, under the condition that the whole input grid is contained in the bounding box of the state. Otherwise, only a subset of the input was parsed. Therefore, a *final state* must satisfy the following conditions:

$$p = \epsilon \rightarrow S$$
$$d = 1$$
$$b = (0,0,n,m)$$

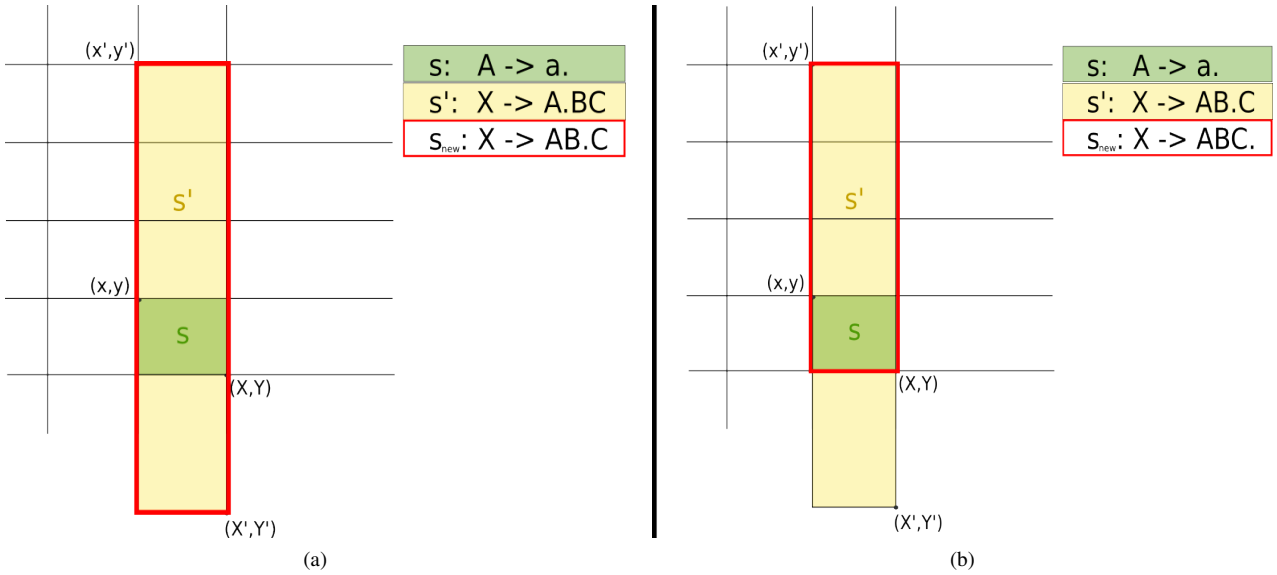

Figure 1: Completion, case 3: (a) Bounding boxes of both complete state s and candidate state s' are aligned. The completed state s_{new} has the same bounding box as s'. (b) State s completes the expansion of non-terminal in state s', therefore limiting the bounding box of the created state s_{new}.

Note that, due to possible grammar ambiguity, there might be multiple final states. Each final state corresponds to a different parse of the input. Furthermore, there is no constraint on the position of the final state in the Earley chart. This contrasts with one-dimensional Earley parsing, where the final state is found only in the last state set of a one-dimensional state chart. Algorithm 1 provides an overview of 2D Earley parsing of an input grid with a given grammar. Functions *Completer, Predictor* and *Scanner* correspond to previously defined operations of the parser.

Algorithm 1 2D Earley parsing

```
function EarleyParse2D (input, grammar)
stateQueue ← ∅
initialState ← [ε → .S (0, 0, n, m)]
push initialState to end of stateQueue
while stateQueue is not empty do
    pop state from front of stateQueue
    if state is complete then
        newState ← Completer(state, grammar)
    else
        symbol Y ← state.p.rhs(d)  //Symbol after the dot
        if Y ∈ N then
            newState ← Predictor(state, grammar)
        else if Y ∈ T then
            newState ← Scanner(state, input, grammar)
    if newState≠ ∅ then
        push newState to stateQueue
end
```

3.2. Remarks

So far, a state was uniquely defined with a tuple $s = (p, d, o, b)$. However, it is also necessary to remember the entire *scanning history* of the state, i.e. to keep track of the symbols in the input grid which were scanned by following the path to the current state. Otherwise, the algorithm may produce two "equal" states in the same Earley chart cell by having scanned different parts of the input grid. One of these states would then overwrite the other, destroying one parsing path completely! Therefore, each state is augmented with a matrix H of the same size as the input grid. This matrix is empty in the initial state.

During **prediction**, the scanning history remains unchanged:

$$s_{new}.H \quad = s.H$$

In the **scanning** step, the new state inherits the history from the old state and updates it with the newly scanned symbol, under the condition that the same position in the grid has not been scanned yet:

$$\begin{aligned} s_{new}.H \quad &= s.H \\ s_{new}.H[i][j] &= a \end{aligned}$$

However, if the same position in grid has already been scanned ($s.H[i][j] \neq \emptyset$), a new state will not be created.

Finally, during **completion**, we make sure that the complete state s and the candidate state s' are "scanning history-compatible":

$$s.H[i][j] = \emptyset \vee s.H[i][j] = \emptyset \vee (s.H[i][j] = s'.H[i][j]) \quad \forall (i, j)$$

If the states are not compatible, completion does not produce a new state. Otherwise, scanning history is propagated from the complete state to the newly created state:

$$s_{new}.H \quad = s.H$$

3.3. An Example

We now demonstrate the algorithm on a simple grammar with 5 productions and a $2 * 2$ input grid, also used in [8]. The grammar contains the following productions:

$$\begin{aligned} p_1 &: S \xrightarrow{H} AA \\ p_2 &: A \xrightarrow{V} BC \\ p_3 &: B \xrightarrow{H} b \\ p_4 &: C \xrightarrow{H} c \\ p_5 &: C \xrightarrow{H} d \end{aligned}$$

where we have used short-hand notations $\xrightarrow{H}$ and $\xrightarrow{V}$ for horizontal and vertical productions, respectively. We omit the scanning matrix H for simplicity. Non-terminal symbols are denoted with capital letters; conversely, terminals are shown in lowercase. S is the grammar axiom. The starting production $\epsilon \rightarrow S$ is added implicitly to the grammar. Note that the grammar is non-deterministic, since a single non-terminal C can be expanded in two different ways. We revisit stochastic productions in Section 4.

We now show, step by step, the complete Earley chart of the parser as it processes the input sequence

$$\begin{matrix} b & b \\ c & d \end{matrix}$$

Obviously, this grid is a valid derivation from the grammar, and there is only a single way to produce it, therefore we expect a single final state. The state chart is seeded with the initial state, corresponding to the starting production.

	0	1	2
0	$\epsilon \xrightarrow{H} .S\ (0, 0, 2, 2)$		
1			
2			

In the prediction step, there is only one state to analyse. Production p_1 is used as the only expansion of non-terminal S, and a new state is added to the same state set.

	0	1	2
0	$\epsilon \xrightarrow{H} .S\ (0,0,2,2)$ $S \xrightarrow{H} .AA\ (0,0,2,2)$		
1			
2			

Prediction continues until there are no more predictions to be made.

	0	1	2
0	$\epsilon \xrightarrow{H} .S\ (0,0,2,2)$ $S \xrightarrow{H} .AA\ (0,0,2,2)$ $A \xrightarrow{V} .BC\ (0,0,2,2)$ $B \xrightarrow{H} .b\ (0,0,2,2)$		
1			
2			

Now, using the last state in the chart in the cell $(0,0)$, the parser scans over the input grid on the position $(1,1)$, i.e. symbol b. This results with a scanned and complete state being added to the appropriate position in the chart.

	0	1	2
0	$\epsilon \xrightarrow{H} .S\ (0,0,2,2)$ $S \xrightarrow{H} .AA\ (0,0,2,2)$ $A \xrightarrow{V} .BC\ (0,0,2,2)$ $B \xrightarrow{H} .b\ (0,0,2,2)$	$B \xrightarrow{H} b.\ (0,0,1,1)$	
1			
2			

Note that the bounding box of the state is adjusted so that it covers only the scanned symbol. Now, the completion step is invoked, as there is a complete state in the current set. The upper-left corner of the complete state is $(0,0)$, so we search for candidate states in that chart cell. The only state that satisfies the conditions for completion is $A \xrightarrow{V} .BC\ (0,0,2,2)$: it has symbol B after the dot, its bounding box encompasses the bounding box of the complete state, and the dot is in the initial position. The complete and candidate state contain horizontal and vertical productions, respectively, so we follow the rules in Section 3.1 and add the completed state in chart cell $(0,1)$.

	0	1	2
0	$\epsilon \xrightarrow{H} .S\ (0,0,2,2)$ $S \xrightarrow{H} .AA\ (0,0,2,2)$ $A \xrightarrow{H} .BC\ (0,0,2,2)$ $B \xrightarrow{H} .b\ (0,0,2,2)$	$B \xrightarrow{H} b.\ (0,0,1,1)$	
1	$A \xrightarrow{V} B.C\ (0,0,1,1)$		
2			

The newly added state passes through the prediction step and creates two predicted states, corresponding to two productions for non-terminal C.

	0	1	2
0	$\epsilon \xrightarrow{H} .S\ (0,0,2,2)$ $S \xrightarrow{H} .AA\ (0,0,2,2)$ $A \xrightarrow{V} .BC\ (0,0,2,2)$ $B \xrightarrow{H} .b\ (0,0,2,2)$	$B \xrightarrow{H} b.\ (0,0,1,1)$	
1	$A \xrightarrow{V} B.C\ (0,0,1,2)$ $C \xrightarrow{H} .c\ (0,1,1,2)$ $C \xrightarrow{H} .d\ (0,1,1,2)$		
2			

The second prediction is discarded during scanning, as the terminal symbol does not match the input. The first prediction, however, creates a new scanned state.

	0	1	2
0	$\epsilon \xrightarrow{H} .S\ (0,0,2,2)$ $S \xrightarrow{H} .AA\ (0,0,2,2)$ $A \xrightarrow{V} .BC\ (0,0,2,2)$ $B \xrightarrow{H} .b\ (0,0,2,2)$	$B \xrightarrow{H} b.\ (0,0,1,1)$	
1	$A \xrightarrow{V} B.C\ (0,0,1,2)$ $C \xrightarrow{H} .c\ (0,1,1,2)$ $C \xrightarrow{H} .d\ (0,1,1,2)$	$C \xrightarrow{H} c.\ (0,1,1,2)$	
2			

The completion step now finishes the derivation of non-terminal A in cell $(0,2)$.

	0	1	2
0	$\epsilon \xrightarrow{H} .S\ (0,0,2,2)$ $S \xrightarrow{H} .AA\ (0,0,2,2)$ $A \xrightarrow{V} .BC\ (0,0,2,2)$ $B \xrightarrow{H} .b\ (0,0,2,2)$	$B \xrightarrow{H} b.\ (0,0,1,1)$	
1	$A \xrightarrow{V} B.C\ (0,0,1,2)$ $C \xrightarrow{H} .c\ (0,1,1,2)$ $C \xrightarrow{H} .d\ (0,1,1,2)$	$C \xrightarrow{H} c.\ (0,1,1,2)$	
2	$A \xrightarrow{V} BC.\ (0,0,1,2)$		

Note that the bounding box of the completed state in cell $(2,0)$ corresponds to scanning the first column of the input grid. This completed state in turn triggers a completion, resulting with the second state in cell $(1,0)$.

	0	1	2
0	$\epsilon \xrightarrow{H} .S\ (0,0,2,2)$ $S \xrightarrow{H} .AA\ (0,0,2,2)$ $A \xrightarrow{V} .BC\ (0,0,2,2)$ $B \xrightarrow{H} .b\ (0,0,2,2)$	$B \xrightarrow{H} b.\ (0,0,1,1)$ $S \xrightarrow{H} A.A\ (0,0,2,2)$	
1	$A \xrightarrow{V} B.C\ (0,0,1,2)$ $C \xrightarrow{H} .c\ (0,1,1,2)$ $C \xrightarrow{H} .d\ (0,1,1,2)$	$C \xrightarrow{H} c.\ (0,1,1,2)$	
2	$A \xrightarrow{V} BC.\ (0,0,1,2)$		

The bounding box of this newest state still encompasses the whole input grid, as there are more non-terminals on the production right-hand side that have not been expanded. Next, this state is used for prediction, creating two new states, one of which is immediately used in the scanning step.

	0	1	2
0	$\epsilon \xrightarrow{H} .S\ (0,0,2,2)$ $S \xrightarrow{H} .AA\ (0,0,2,2)$ $A \xrightarrow{V} .BC\ (0,0,2,2)$ $B \xrightarrow{H} .b\ (0,0,2,2)$	$B \xrightarrow{H} b.\ (0,0,1,1)$ $S \xrightarrow{H} A.A\ (0,0,2,2)$ $A \xrightarrow{V} .BC\ (1,0,2,2)$ $B \xrightarrow{H} .b\ (1,0,2,2)$	$B \xrightarrow{H} b.\ (1,0,2,1)$
1	$A \xrightarrow{V} B.C\ (0,0,1,2)$ $C \xrightarrow{H} .c\ (0,1,1,2)$ $C \xrightarrow{H} .d\ (0,1,1,2)$	$C \xrightarrow{H} c.\ (0,1,1,2)$	
2	$A \xrightarrow{V} BC.\ (0,0,1,2)$		

After the completion and prediction steps:

	0	1	2
0	$\epsilon \xrightarrow{H} .S\ (0,0,2,2)$ $S \xrightarrow{H} .AA\ (0,0,2,2)$ $A \xrightarrow{V} .BC\ (0,0,2,2)$ $B \xrightarrow{H} .b\ (0,0,2,2)$	$B \xrightarrow{H} b.\ (0,0,1,1)$ $S \xrightarrow{H} A.A\ (0,0,2,2)$ $A \xrightarrow{V} .BC\ (1,0,2,2)$ $B \xrightarrow{H} .b\ (1,0,2,2)$	$B \xrightarrow{H} b.\ (1,0,2,1)$
1	$A \xrightarrow{V} B.C\ (0,0,1,2)$ $C \xrightarrow{H} .c\ (0,1,1,2)$ $C \xrightarrow{H} .d\ (0,1,1,2)$	$C \xrightarrow{H} c.\ (0,1,1,2)$ $A \xrightarrow{V} B.C\ (1,0,2,2)$ $C \xrightarrow{H} .c\ (1,1,2,2)$ $C \xrightarrow{H} .d\ (1,1,2,2)$	
2	$A \xrightarrow{V} BC.\ (0,0,1,2)$		

The last symbol in the grid is subsequently scanned, which triggers a recursive completion step, resulting with a final state.

	0	1	2
0	$\epsilon \xrightarrow{H} .S\ (0,0,2,2)$ $S \xrightarrow{H} .AA\ (0,0,2,2)$ $A \xrightarrow{V} .BC\ (0,0,2,2)$ $B \xrightarrow{H} .b\ (0,0,2,2)$	$B \xrightarrow{H} b.\ (0,0,1,1)$ $S \xrightarrow{H} A.A\ (0,0,2,2)$ $A \xrightarrow{V} .BC\ (1,0,2,2)$ $B \xrightarrow{H} .b\ (1,0,2,2)$	$B \xrightarrow{H} b.\ (1,0,2,1)$ $S \xrightarrow{H} AA.\ (0,0,2,2)$ $\epsilon \xrightarrow{\mathrm{H}} \mathrm{S}.\ (0,0,2,2)$
1	$A \xrightarrow{V} B.C\ (0,0,1,2)$ $C \xrightarrow{H} .c\ (0,1,1,2)$ $C \xrightarrow{H} .d\ (0,1,1,2)$	$C \xrightarrow{H} c.\ (0,1,1,2)$ $A \xrightarrow{V} B.C\ (1,0,2,2)$ $C \xrightarrow{H} .c\ (1,1,2,2)$ $C \xrightarrow{H} .d\ (1,1,2,2)$	$C \xrightarrow{H} d.\ (1,1,2,2)$
2	$A \xrightarrow{V} BC.\ (0,0,1,2)$	$A \xrightarrow{V} BC.\ (1,0,2,2)$	

The state highlighted in green satisfies all final state conditions, which means that the algorithm has found a valid parse of the input. In contrast, by following the method described in [8], the final parse chart would be as follows:

	0	1	2
0	$\epsilon \xrightarrow{H} .S\ (0,0,2,2)$ $S \xrightarrow{H} .AA\ (0,0,2,2)$ $A \xrightarrow{V} .BC\ (0,0,2,2)$ $B \xrightarrow{H} .b\ (0,0,2,2)$	$B \xrightarrow{H} b.\ (0,0,2,1)$	$S \xrightarrow{H} A.A\ (0,0,2,2)$ $A \xrightarrow{V} .BC\ (2,0,2,2)$ $B \xrightarrow{H} .b\ (2,0,2,2)$
1	$A \xrightarrow{V} B.C\ (0,0,2,2)$ $C \xrightarrow{H} .c\ (0,1,2,2)$ $C \xrightarrow{H} .d\ (0,1,2,2)$	$C \xrightarrow{H} c.\ (0,1,2,2)$	
2	$A \xrightarrow{V} BC.\ (0,0,2,2)$		

Notice the different bounding box of non-terminal B in chart position $(1,0)$ which propagates to subsequent states, making it impossible to parse the input. Since Figure 18.3 in [8] shows a successful parse of this input, we can only conclude there is an omission in their algorithm, likely in the scanning step. However, even if the scanning step of their approach is fixed to calculate the correct bounding box, the algorithm of [8] does not properly handle cases such as the one exemplified in Figure 1. We demonstrate this on a more challenging input, shown in Figure 2.

b b c d e e	$p_1: \quad S \xrightarrow{V} X_1X_2$ $p_2: \quad X_1 \xrightarrow{H} AA$ $p_3: \quad X_2 \xrightarrow{H} EE$ $p_4: \quad A \xrightarrow{V} BC$ $p_5: \quad B \xrightarrow{H} b$ $p_6: \quad C \xrightarrow{H} c$ $p_7: \quad C \xrightarrow{H} d$ $p_8: \quad E \xrightarrow{H} e$

Figure 2: A more difficult example. Left: Input grid, right: input grammar

We parse the input with both the approach of [8] (after fixing the scanning step) and our approach, and show the final Earley charts in Figure 3. Our approach successfully parses the input with the given grammar, while the approach of [8] fails due to absence of X'' and Y'' terms in the completion step.

	0	1	2
0	$\epsilon \xrightarrow{H} .S\ (0,0,2,3)$ $S \xrightarrow{V} .X_1X_2\ (0,0,2,3)$ $X_1 \xrightarrow{H} .AA\ (0,0,2,3)$ $A \xrightarrow{V} .BC\ (0,0,2,3)$ $B \xrightarrow{H} .b\ (0,0,2,3)$	$B \xrightarrow{H} b.\ (0,0,1,1)$ $X_1 \xrightarrow{H} A.A\ (0,0,2,3)$ $A \xrightarrow{V} .BC\ (1,0,2,3)$ $B \xrightarrow{H} .b\ (1,0,2,3)$	$B \xrightarrow{H} b.\ (1,0,2,1)$ $X_1 \xrightarrow{H} AA.\ (0,0,2,3)$
1	$A \xrightarrow{V} B.C\ (0,0,1,3)$ $C \xrightarrow{H} .c\ (0,1,1,3)$ $C \xrightarrow{H} .d\ (0,1,1,3)$	$C \xrightarrow{H} c.\ (0,1,1,2)$ $A \xrightarrow{V} B.C\ (1,0,2,3)$ $C \xrightarrow{H} .c\ (1,1,2,3)$ $C \xrightarrow{H} .d\ (1,1,2,3)$	$C \xrightarrow{H} d.\ (1,1,2,2)$
2	$A \xrightarrow{V} BC.\ (0,0,1,3)$	$A \xrightarrow{V} BC.\ (1,0,2,3)$	
3	$S \xrightarrow{V} X_1.X_2\ (0,0,2,3)$ $X_2 \xrightarrow{H} .EE\ (0,3,2,3)$ $E \xrightarrow{H} .e\ (0,3,2,3)$		

(a)

	0	1	2
0	$\epsilon \xrightarrow{H} .S\ (0,0,2,3)$ $S \xrightarrow{V} .X_1X_2\ (0,0,2,3)$ $X_1 \xrightarrow{H} .AA\ (0,0,2,3)$ $A \xrightarrow{V} .BC\ (0,0,2,3)$ $B \xrightarrow{H} .b\ (0,0,2,3)$	$B \xrightarrow{H} b.\ (0,0,1,1)$ $X_1 \xrightarrow{H} A.A\ (0,0,2,2)$ $A \xrightarrow{V} .BC\ (1,0,2,2)$ $B \xrightarrow{H} .b\ (1,0,2,2)$	$B \xrightarrow{H} b.\ (1,0,2,1)$ $X_1 \xrightarrow{H} AA.\ (0,0,2,2)$
1	$A \xrightarrow{V} B.C\ (0,0,1,3)$ $C \xrightarrow{H} .c\ (0,1,1,3)$ $C \xrightarrow{H} .d\ (0,1,1,3)$	$C \xrightarrow{H} c.\ (0,1,1,2)$ $A \xrightarrow{V} B.C\ (1,0,2,2)$ $C \xrightarrow{H} .c\ (1,1,2,2)$ $C \xrightarrow{H} .d\ (1,1,2,2)$	$C \xrightarrow{H} d.\ (1,1,2,2)$
2	$A \xrightarrow{V} BC.\ (0,0,1,2)$ $S \xrightarrow{V} X_1.X_2\ (0,0,2,3)$ $X_2 \xrightarrow{H} .EE\ (0,2,2,3)$ $E \xrightarrow{H} .e\ (0,2,2,3)$	$A \xrightarrow{V} BC.\ (1,0,2,2)$ $E \xrightarrow{H} e.\ (0,2,1,3)$ $X_2 \xrightarrow{H} E.E\ (0,2,2,3)$ $E \xrightarrow{H} .e\ (1,2,2,3)$	$E \xrightarrow{H} e.\ (1,2,2,3)$ $X_2 \xrightarrow{H} EE.\ (0,2,2,3)$
3	$S \xrightarrow{V} X_1X_2.\ (0,0,2,3)$ $\epsilon \xrightarrow{H} S.\ (0,0,2,3)$		

(b)

Figure 3: Parsing the example from Figure 2: a) Parsing chart of [8] with fixed scanning step. No final states exist in the chart. b) Our approach. The input is correctly parsed.

4. Extension to Stochastic Grammars

As said in Section 2, in a stochastic context free grammar (SCFG), every production contains an associated probability. Consequently, the grammar generates each output derivation with an associated probability. Due to ambiguities in the grammar, there might be multiple ways of generating a single output. As defined previously, the likelihood of a grammar G generating a lattice l is defined as $L(l|G) = \sum_{\delta \Rightarrow l} P(\delta)$, summed over all derivations that yield a particular lattice (grid). However, among these alternative derivations, we are interested in the *most-likely path,* i.e. the sequence of rules associated with the maximum probability. This sequence is called the *Viterbi parse,* and its corresponding probability the *Viterbi probability*.

Our 2D Earley parser can be easily adapted to calculate the Viterbi parse of a given derivation, by following the procedure described in [7]. Essentially, every state in the Earley chart is assigned a Viterbi probability. These probabilities are then propagated and updated during parsing. Finally, the derivation probability is determined by reading out the Viterbi probabilities in all final states in the chart, and selecting the maximum value.

To describe the probability calculation algorithm, the concept of *predecessor* states needs to be introduced. As described in Section 3, new states are created and added to the chart based on previous states. In prediction and scanning, there is only one predecessor state, while in completion step the new state is created based on two predecessor states. Therefore, for each state, we can keep track of its predecessor(s).

Calculation of Viterbi probabilities is performed as follows. First, the initial state is assigned the Viterbi probability of 1. Subsequently, when a new state is being added to the chart, its probability is calculated depending on the parser operation:

Prediction

For a predecessor state s with Viterbi probability of p:

$$(i, j)\ A \rightarrow \lambda.B\mu \quad [p]$$

the Viterbi probability of every successor state s' depends solely on the probability of the production in state s:

$$(i, j)\ B \rightarrow .\nu \quad [P(A \rightarrow \lambda B\mu)]$$

Scanning

Since the terminal symbol was already selected during prediction, Viterbi probability simply propagates to the new state. The predecessor state s:

$$(i, j)\ A \rightarrow .a \quad [p]$$

results in the successor state s':

$$(i + 1, j)\ A \rightarrow a. \quad [p]$$

Completion

Every state s'' produced by completion will have two predecessors s and s':

$$(i, j)\ B \rightarrow \nu. \quad [p]$$

$$(x, y)\ A \rightarrow \lambda.B\mu \quad [p']$$

Its Viterbi probability is then calculated by multiplying the probabilities of predecessors:

$$(i, j)\ A \rightarrow \lambda B.\mu \quad [pp']$$

However, the same state (and with the same scanning history) might be generated multiple times, because there can be several plausible input parses. Therefore, if a state already exists in a chart, its Viterbi probability and predecessor list must be updated. Since we are interested only in the parsing path of maximum probability, we replace the existing state with the new state only if the new Viterbi probability exceeds the current highest value.

4.1. Parser Output

After the parsing has finished, we search the Earley chart for final states. If none are found, the input cannot be parsed with the current grammar. Otherwise, we select the final state with the highest Viterbi probability, and return that value as the probability of the input grid.

Often, one also needs to know the exact productions of the grammar used to parse the input, and their count. These *expected usage counts* can be used, for example, to find the maximum likelihood production probabilities based on a set of training instances [6, 7], using an expectation-maximization algorithm. The productions used in the Viterbi parse of the input can be extracted from the Earley chart, by tracing back the path from the final state to the initial state. The outline of the procedure is shown in Algorithm 2.

Algorithm 2 Calculating expected production counts by backtracking in Earley chart

```
function Backtrack (state, productionCounts)
if state.d=0 then
    // State generated by prediction
    increment(productionCounts[state.p])
else
    if state.p.rhs(d)∈T then
        // State generated by scanning
        productionCounts ← ViterbiBacktrack (state.predecessor, productionCounts)
    else
        //State generated by completion
        productionCounts ← ViterbiBacktrack (state.predecessor, productionCounts)
        productionCounts ← ViterbiBacktrack (state.predecessor2, productionCounts)
return productionCounts
```

The recursive function *Backtrack* is called with the final state and production counts initialized to zero. The algorithm is essentially a depth-first search, since states generated by completion provide two separate paths to follow. The recursion stops whenever a state generated by prediction is encountered. At this point, the usage count of the corresponding production is incremented. When the algorithm finishes, the variable *productionCounts* will contain the desired results. For instance, the output for example in Figure 2 is $[1\,1\,1\,2\,2\,1\,1\,2]$.

5. Conclusion

In this paper, we have presented a detailed description of an Earley parsing algorithm for two-dimensional stochastic context free grammars. To the best of our knowledge, this is the first algorithm that combines the insights from both deterministic multidimensional grammars parsing and one-dimensional stochastic grammar parsing in a top-down approach. Additionally, we have demonstrated our approach on two toy examples of two-dimensional grid parsing. Finally, this algorithm was successfully used in the field of inverse procedural modeling, for the task of parsing building facades. For more information, please see [6].

Acknowledgement

This work was supported by ERC Advanced Grant VarCity and Research Programme of the Fund for Scientific Research-Flanders (Belgium) (FWO - G.0004.08).

References

[1] F. Alvaro, J.-A. Sánchez, and J.-M. Benedí. Recognition of printed mathematical expressions using two-dimensional stochastic context-free grammars. In *ICDAR*, 2011. 1

[2] G. Costagliola and S.-K. Chang. Using linear positional grammars for the LR parsing of 2-d symbolic languages. *Grammars*, 2(1):1–34, 1999. 1

[3] C. Crimi, A. Guercio, G. Nota, G. Pacini, G. Tortora, and M. Tucci. Relation grammars and their application to multi-dimensional languages. *J. Vis. Lang. Comput.*, 2(4):333–346, Dec. 1991. 1

[4] J. Earley. An efficient context-free parsing algorithm. *Commun. ACM*, 13(2), Feb. 1970. 1

[5] M. Lange and H. Leiß. To CNF or not to CNF? An efficient yet presentable version of the CYK algorithm. *Informatica Didactica*, 8, 2009. 1

[6] A. Martinović and L. Van Gool. Bayesian Grammar Learning for Inverse Procedural Modeling. In *CVPR*, 2013. To appear. 1, 13

[7] A. Stolcke. *Bayesian Learning of Probabilistic Language Models*. PhD thesis, University of California at Berkeley, 1994. 1, 2, 3, 12, 13

[8] M. Tomita. Parsing 2-dimensional language. In M. Tomita, editor, *Current Issues in Parsing Technology*, volume 126 of *The Springer International Series in Engineering and Computer Science*, pages 277–289. Springer US, 1991. 1, 2, 6, 9, 10, 11

[9] S. Wild. An earley-style parser for solving the rna-rna interaction problem (bachelor thesis), 2010. 1

[10] D. H. Younger. Recognition and parsing of context-free languages in time n3. *Information and Control*, 10(2), 1967. 1