Choosing Word Occurrences for the Smallest Grammar Problem*

Rafael Carrascosa[1], François Coste[2], Matthias Gallé[2], and Gabriel Infante-Lopez[1,3]

[1] Grupo de Procesamiento de Lenguaje Natural
Universidad Nacional de Córdoba, Argentina
[2] Symbiose Project
IRISA/INRIA Rennes-Bretagne Atlantique, France
[3] Consejo Nacional de Investigaciones Científicas, Argentina

Abstract. The smallest grammar problem - namely, finding a smallest context-free grammar that generates exactly one sequence - is of practical and theoretical importance in fields such as Kolmogorov complexity, data compression and pattern discovery. We propose to focus on the choice of the occurrences to be rewritten by non-terminals. We extend classical offline algorithms by introducing a global optimization of this choice at each step of the algorithm. This approach allows us to define the search space of a smallest grammar by separating the choice of the non-terminals and the choice of their occurrences. We propose a second algorithm that performs a broader exploration by allowing the removal of useless words that were chosen previously. Experiments on a classical benchmark show that our algorithms consistently find smaller grammars then state-of-the-art algorithms.

1 Introduction

The smallest grammar problem - namely, finding a smallest context-free grammar that generates exactly one sequence - is of practical and theoretical importance in fields such as Kolmogorov complexity, data compression and pattern discovery.

The size of a smallest grammar can be considered a computable variant of Kolmogorov complexity, in which the Turing machine description of the sequence is restricted to context-free grammars. The problem is then decidable, but still hard: the problem of finding a smallest grammar with an approximation ratio smaller then $\frac{8569}{8568}$ is NP-HARD [1]. Nevertheless, an $O(\log^3 n)$ approximation ratio - with n the length of the sequence - can be achieved by a simple algorithmic scheme based on approximation to the shortest superstring problem [1] and a smaller $O(\log n/g)$ (where g is the size of a smallest grammar) approximation ratio is possible by more complex mappings from the LZ77-factorization of the sequence to a context-free grammar with a balanced parsing tree [1, 2].

* The work described in this paper is partially supported by the Program of International Scientific Cooperation MINCyT - INRIA/CNRS

If the grammar is small, storing the grammar instead of the sequence can be interesting from a data compression perspective. Kieffer and Yang developed the formal framework of compression by *Grammar Based Codes* from the viewpoint of information theory, defining irreducible grammars and demonstrating their universality [3]. Before this formalization, several algorithms allowing to compress a sequence by context-free grammars had already been proposed. The LZ78-factorization introduced by Ziv and Lempel in [4] can be interpreted as a context-free grammar. Let us remark that this is not true for LZ77, published one year before [5]. Moreover, it is a commonly used result that the size of a LZ77-factorization is a lower bound on the size of a smallest grammar [2, 1]. The first approach that generated explicitly a context-free grammar with compression ability is *Sequitur* [6]. Like LZ77 and LZ78, Sequitur is an on-line algorithm that processes the sequence from left to right. It maintains incrementally a grammar generating the part of the sequence read, introducing and deleting rewriting rules to ensure that no digram (pair of adjacent symbols) occurs more than once and that each rule is used at least twice. Other algorithms consider the entire sequence before choosing which repeat will be rewritten by the introduction of a new rule. Most of these offline algorithms proceed in a greedy manner. First the grammar is initialized by a unique initial rule $S \rightarrow s$ where s is the input sequence. Then they proceed iteratively, selecting in each iteration one repeated word w according to a score function and replacing all the (non-overlapping) occurrences of the repeat w in the whole grammar by a new terminal N and adding the new rewriting rule $N \rightarrow w$ to the grammar. Different heuristics have been used to choose the repeat: Frequent [7] chooses the most frequently-occurring digram, Long [8] chooses the longest word while Compressive [9] chooses the word that reduce at most the size of the resulting grammar. Greedy [10] belongs also to this family but the score used for choosing the words is oriented toward directly optimizing the number of bits needed to encode the grammar rather than minimizing its size. The running time of Sequitur is linear and linear-time versions of Frequent and Long have been introduced in [11] and [12] respectively, while the existence of a linear-time algorithm for Compressive and Greedy remains an open question.

In pattern discovery, a smallest grammar is a good candidate for being the one that generates the data according to Occam's razor principle. In that case, the grammar may not only be used for compressing the sequence but also to unveil its structure. Inference of the hierarchical structure of sequences was the initial motivation of Sequitur and has been the subject of several papers applying this scheme to DNA sequences [6, 13, 14], musical scores [15] or natural language [7, 16]. It can also be a first step to learn more general grammars along the lines of [17]. In all the latter cases, a slight difference in the size of the grammar, which would not matter for data compression, can dramatically change the results with respect to the structure and more sophisticated algorithms than those for data compression are needed. In this article, we focus on how to choose occurrences that are going to be rewritten. This mechanism is generally handled straightforwardly in these papers and consists of selecting *all* the non-overlapping

occurrences in a left to right order. Moreover, once an occurrence has been chosen for being rewritten, the result is definitive and is not altered by the words that will be chosen in the following iterations. In order to remedy these flaws, we show how to globally optimize the choice of the occurrences to be replaced by non-terminals. We are then able to improve classical greedy algorithms by introducing this optimization step at each iteration of the algorithm. This optimization allows us to separate the choice of the non-terminals and the choice of their occurrences. We redefine the search space and we propose a new procedure performing a wider search by adding the possibility to discard non-terminals previously included in the grammar.

The outline of this paper is the following: in Sect. 2 we introduce formally the definitions and the classical offline algorithms. Section 3 contains our contributions: in Sect. 3.1 we show how to optimize the choice of occurrences to be replaced by non-terminals for a set of words and extend offline algorithms by optimizing the choice of the occurrences at each step. We show that this optimization can also be used directly to guide the search in a new algorithm in Sect. 3.3. We present experiments on a classical benchmark in Sect. 4 showing that the occurrence optimization consistently allows to find smaller grammars and Sect. 5 concludes the paper.

2 Iterative Repeat Replacement Algorithms

2.1 Definitions and Notation

We start by giving a few definitions and setting up the nomenclature that we use along the paper. A string s is a sequence of characters $s_1 \dots s_n$, its length, $|s| = n$. ϵ denotes the empty word, and $s[i:j] = s_i \dots s_j$, $s[i:j] = \epsilon$ if $j < i$. A context-free grammar is a tuple $\langle \Sigma, \mathcal{N}, \mathcal{P}, S \rangle$, where Σ is the set of *terminals*, $\mathcal{N}$ is the set of *non-terminals* and $\mathcal{N}$ and Σ are disjoint. $S \in \mathcal{N}$ is called the *start symbol* and $\mathcal{P}$ is a set of *productions*. Each production is of the form $A \to \alpha$ where its *left-hand side* A is a non-terminal and its *right-hand side* α belongs to $(\Sigma \cup \mathcal{N})^*$. β can be derived from α, denoted by $\alpha \Rightarrow \beta$, if there exists a set of production rules allowing to produce β starting from α. The language defined by a grammar is the set of words $\{w \in \Sigma^* : S \Rightarrow w\}$.

Several definitions of the grammar size exist. Following [9], we define the *size of the grammar* G, denoted by $|G|$, to be the length of its encoding by concatenation of its right-hand sides separated by end-of-rule markers: $|G| = \sum_{A \to \alpha \in \mathcal{P}} (|\alpha| + 1)$.

2.2 General Scheme

Most offline algorithms follow the same general scheme. First, the grammar is initialized with a unique initial rule $S \to s$ where s is the input sequence and then they proceed iteratively. At each iteration, a word ω occurring more than once in s is chosen according to a score function f, all the (non-overlapping)

occurrences of ω in the grammar are replaced by a new non-terminal N_ω and a new rewriting rule $N_\omega \rightarrow \omega$ is added to the grammar. We give pseudo-code for this general scheme that we name *Iterative Repeat Replacement* (IRR) in Algorithm 1. There, $\mathcal{P}$ is the set of production rules being built: this defines a unique grammar $G(\mathcal{P})$ and therefore we define $|\mathcal{P}| = |G(\mathcal{P})|$. The set of repeated words in the right-hand side of $\mathcal{P}$ is denoted by repeats($\mathcal{P}$) and $\mathcal{P}_{\omega \mapsto N}$ is the result of the substitution of ω by the new symbol N in the right-hand sides of $\mathcal{P}$ as detailed in the next paragraph.

When occurrences overlap, one has to specify which occurrences have to be replaced. One solution is to choose all the elements in the *canonical list of non-overlapping occurrences* of ω in s, which we define to be the list of non-overlapping occurrences of ω in a greedy left to right way (all occurrences overlapping with the first selected occurrence are not considered, then the same thing with the second non-eliminated occurrence, etc). This ensures that a maximal number of occurrences will be replaced. When searching for the smallest grammar, one has to consider not only the occurrences of a word in s but also their occurrence in right-hand sides of rules that are currently part of the grammar. A canonical list of non-overlapping occurrences of ω can be defined for each right-hand side appearing in the set of production rules $\mathcal{P}$. This provides a straightforward list of occurrences used in the scoring function or the replacement step by our pseudo-code defining IRR: $\mathcal{P}_{\omega \mapsto N}$ denotes the result of substituting by N these right-hand side occurrences of ω in $\mathcal{P}$.

Algorithm 1 Iterative Repeat Replacement

IRR(s, f)
Require: s is a sequence, and f is a score function on words
1: $\mathcal{N} \leftarrow \{N_s\}$
2: $\mathcal{P} \leftarrow \{N_s \rightarrow s\}$
3: **while** $\exists \omega \, : \, \omega \leftarrow \arg\max_{\alpha \in \text{repeats}(\mathcal{P})} f(\alpha, \mathcal{P}) \wedge |\mathcal{P}_{\omega \mapsto N_\omega}| < |\mathcal{P}|$ **do**
4: $\quad \mathcal{N} \leftarrow \mathcal{N} \cup \{N_\omega\}$
5: $\quad \mathcal{P} \leftarrow \mathcal{P}_{\omega \mapsto N_\omega} \cup \{N_\omega \rightarrow \omega\}$
6: **end while**
7: **return** $G(\mathcal{P})$

The IRR scheme enables us to compare in a uniform framework the behavior of different score functions f that are used in the classical algorithms for choosing the words to replace. We implemented the scores of the most popular offline algorithms (or their extension to include right-hand sides, if that was not considered originally). IRR-MO maximizes the number of non-overlapping occurrences, it uses $f(\alpha, \mathcal{P}) = o$, where o is the size of the canonical non-overlapping list of α in the right-hand sides of rules in $\mathcal{P}$ (Frequent [7] or Re-Pair [11]); IRR-ML selects the largest repeated word: $f(\alpha, \mathcal{P}) = |\alpha|$ (Long [8] LFS, or LFS2 [12]); and IRR-MC minimizes the size of the grammar by maximizing $f(\alpha, \mathcal{P}) = o * |\alpha| - o - |\alpha| - 1$ (Compressive [9]). The complexity of IRR when

it uses one of these scores is $O(n^3)$ since for a sequence of size n, the computation of the scores involving only o and $|\alpha|$ of the $O(n^2)$ possible repeats can be done in $O(n^2)$ using a suffix tree structure and the number of iterations is bounded by n since the size of the grammar decreases at each step.

The grammars found by these algorithms, Sequitur and LZ78 on a small example are shown in Fig. 1 while a comparison of the size of the grammars returned by these algorithms over a standard data compression corpus are presented in Sect. 4. These experiments confirm that IRR-MC is the best of these practical heuristics for finding smaller grammars as was suggested in [9]. Even when theoretical algorithms were designed to achieve a low approximation ratio [1, 2, 18], until now it could not be proven (theoretically or empirically) that they perform better than IRR-MC.

$S \rightarrow N_1\mathrm{dabge}N_1\mathrm{e}N_1\mathrm{d\$}$
$N_1 \rightarrow \mathrm{a\ b\ c}$

IRR-MC

$S \rightarrow N_2\mathrm{d}N_1\mathrm{ge}N_2\mathrm{e}N_2\mathrm{d\$}$
$N_1 \rightarrow \mathrm{a\ b}$
$N_2 \rightarrow N_1\ \mathrm{c}$

IRR-MO

$S \rightarrow N_1\mathrm{abge}N_2\mathrm{e}N_1\mathrm{\$}$
$N_1 \rightarrow N_2\ \mathrm{d}$
$N_2 \rightarrow \mathrm{a\ b\ c}$

IRR-ML

$S \rightarrow N_1\mathrm{d}N_2\mathrm{g}N_3N_3\mathrm{d\$}$
$N_1 \rightarrow N_2\ \mathrm{c}$
$N_2 \rightarrow \mathrm{a\ b}$
$N_3 \rightarrow \mathrm{e}\ N_1$

Sequitur

$S \rightarrow N_1N_2N_3N_4N_5N_6N_7N_8N_9N_{10}N_{11}$
$N_1 \rightarrow \mathrm{a}$
$N_2 \rightarrow \mathrm{b}$
$N_3 \rightarrow \mathrm{c}$
$N_4 \rightarrow \mathrm{d}$
$N_5 \rightarrow N_1\ \mathrm{b}$
$N_6 \rightarrow \mathrm{g}$
$N_7 \rightarrow \mathrm{e}$
$N_8 \rightarrow N_5\ \mathrm{c}$
$N_9 \rightarrow N_7\ \mathrm{a}$
$N_{10} \rightarrow N_2\ \mathrm{c}$
$N_{11} \rightarrow \mathrm{d\ \$}$

LZ78

Fig. 1. Grammars returned by classical algorithms on sequence *abcdabgeabceabcd$*

2.3 Limitations of IRR

Even though IRR algorithms are the best known practical algorithms for obtaining small grammars, they present some weaknesses. In the first place, their greedy strategy does not guarantee that the compression gain introduced by a selected word ω will still be interesting in the final grammar. Each time a future iteration selects a substring of ω, the length of the production rule is reduced; and each time a superstring is selected, its number of occurrences is reduced. Moreover, the first choices mark some breaking points and future words may appear inside them or in another parts of the grammar, but never cross them.

One could argue that it could be possible to find a score function that considers probable choices in the future. Nevertheless, a third weakness is intrinsic to IRR and does not depend on the score function: consider the sequence $xaxbxcx\#_1xbxcxax\#_2xcxaxbx\#_3xaxcxbx\#_4xbxaxcx\#_5xcxbxax\#_6xax\#_7xbx\#_8xcx$,

where each $\#_i$ acts as a separator over which no repeat spans. This sequence exploits the fact that IRR algorithms replace all possible occurrences of the selected word. Let us define G^* as the following grammar:

$$S \rightarrow AbC\#_1BcA\#_2CaB\#_3AcB\#_4BaC\#_5CbA\#_6A\#_7B\#_8C$$
$$A \rightarrow xax \quad B \rightarrow xbx \quad C \rightarrow xcx$$

$|G^*| = 42$. Note that no IRR algorithm could generate G^* and, moreover, the smallest possible grammar that can be obtained with an IRR algorithm has size 46, resulting in an approximation ratio of 1.095. This is a general lower bound for *any* IRR algorithm.

In order to find G^*, the choice of occurrences that will be rewritten should be flexible when considering repeats introduced in future iterations.

3 Optimization of the occurrences choice

3.1 Global Optimization of Occurrences Replacement

Once an IRR algorithm has chosen a repeated word ω, it replaces all non-overlapping occurrences of that word in the current grammar by a new non-terminal N and then adds $N \rightarrow \omega$ to the set of production rules. In this section, we propose to perform a global optimization of the replacement of occurrences, considering not only the last non-terminal but also all the previously introduced non-terminals. The idea is to allow occurrences of words to be kept (instead of being replaced by non-terminals) if replacing other occurrences of words overlapping them results in a better compression.

Let $\mathcal{N}$ denote the set of non-terminals that can be used for replacing occurrences. Let us remark that each non-terminal N introduced to replace a word ω is not limited to replace ω but can replace any word with the same yield where we define the *yield* of a word $\alpha \in (\Sigma \cup N)^*$ by: $yield(\alpha) = \{w \in \Sigma^* / \alpha \Rightarrow w\}$. For instance, given the set of non-terminals $\mathcal{N}$ and their respective yields (defined at the moment of the introduction of the rule $N \rightarrow \alpha$ by $yield_N \leftarrow yield(\alpha)$), we can search for the best replacement in the sequence s by non-terminals of $\mathcal{N} \setminus \{S\}$ such that the replacement results in a minimal sequence s' and the yield of s' is s. This result would provide us with a minimal rule $N_s \rightarrow s'$ producing s and assuming the production of their yield by other non-terminals, which can also in turn be minimized by the same kind of optimization. This problem is related to the problem of static dictionary parsing [19] with the difference that the dictionary also has to be parsed. It can be formalized here as searching for the smallest grammar with a set of production rules of the form $\{N \rightarrow \alpha / N \in \mathcal{N}, \alpha \in (\mathcal{N} \cup \Sigma)^*, yield(\alpha) = yield_N\}$, the set of non-terminals $\mathcal{N}$ and their respective yields being given.

This problem can be solved in a classical way by searching for the shortest path in $|\mathcal{N}|$ graphs as follows. For each non-terminal $N \in \mathcal{N}$, we introduce a directed labeled acyclic graph Γ_N. To lighten the notation, we assume that the yield of N can be written as $yield_N = y_1 \ldots y_k$. Then, we define the graph Γ_N to

have $k+1$ nodes, namely $\{1 \ldots k+1\}$, and edge from node i to node $i+1$ labeled with y_i for each i, and an edge from node i to $j+1$ labeled by M if there exists a non-terminal M different from N such that $y[i:j] = yield_M$. Intuitively, an edge from node i to node $j+1$ with label M represents a possible replacement of the occurrence $y[i:j]$ by M. Searching for the smallest path from state 1 to state $k+1$ with a classical dynamic programming algorithm allows us to find the smallest α such that $\alpha \Rightarrow yield_N$. This procedure is done for each non-terminal. We denote hereafter $\mathcal{P}_{min}(\mathcal{N})$ the set of rules obtained by this optimization.

3.2 IRR with Occurrence Optimization

We can now define the variant of IRR, called *Iterative Repeat Choice with Occurrence Optimization* (IRCOO) with the pseudo-code given in Algorithm 2. The smallest path algorithm has complexity $O(k \times m)$ for a graph Γ_N, where k is the number of nodes of Γ_N $(= |yield(N)|)$ and $m = |\mathcal{N}|$. k is bounded by $|s| = n$, so the complexity of computing $\mathcal{P}_{min}$ is $O(n \times m^2)$. The computation of the $\arg\max$ depends only on the number of repeats, assuming that f is constant, so that its complexity lies in $O(n^2)$. Like for IRR, the total number of times the while loop is executed is bounded by n. The complexity of this generic scheme is thus $O(n \times (n^2 + n \times m^2))$.

Algorithm 2 Iterative Repeat Choice with Occurrences Optimization

IRCOO(s, f)

Require: s is a sequence, and f is a score function on words

1: $\mathcal{N} \leftarrow \{N_s\}$
2: $\mathcal{P} \leftarrow \{N_s \rightarrow s\}$
3: **while** $(\exists \omega \,:\, \omega \leftarrow \text{argmax}_{\alpha \in \text{repeats}(\mathcal{P})} f(\alpha, \mathcal{P})) \wedge \left|\mathcal{P}_{min}(\mathcal{N} \cup \{N_{yield(\omega)}\})\right| < |\mathcal{P}|$ **do**
4: $\quad \mathcal{N} \leftarrow \mathcal{N} \cup \{N_{yield(\omega)}\}$
5: $\quad \mathcal{P} \leftarrow \mathcal{P}_{min}(\mathcal{N})$
6: **end while**
7: **return** $G(\mathcal{P})$

As an example, consider again the sequence from Sect. 2.3. After three iterations of IRCOO-MC the words xax, xbx and xcx are chosen, and the $\mathcal{P}_{min}$ of these non-terminals and the original sequence results in G^*.

IRRCOO extends IRR by performing a global optimization at each step of the replaced occurrences but still relies on the classical score functions of IRR to choose the words to introduce. But the result of the optimization can be used directly to guide the search in a hill-climbing like approach that we introduce in the next subsection.

3.3 Widening the Explored Space: The ZZ Algorithm

In this section we divert from IRR algorithms by taking the idea presented in IRCOO a step forward. In the optimization of the occurrences replacement performed in Sect. 3.1, the choice of the non-terminals implicitly implied their yields: there was a direct relation between non-terminals and terminals, but the focus was on non-terminals. Instead, we can focus on yields and then, the optimization algorithm can be seen as a procedure that takes a string s and a set of its substrings as input and that returns the smallest grammar that can be built from it, provided that the grammar produces s and that, for each substring in the input set, there exists a non-terminal in the grammar whose yield is the substring itself. The optimization procedure works as a scoring function for sets of substrings: the size of the grammar produced by the optimization procedure is the score of the given set of substrings.

In this section, we take advantage of this idea and present an algorithm, called *Zig Zag* (ZZ), that traverses in a hill-climbing way the lattice of the subsets of repeated substrings of the string s. The search space is a lattice that has one node for each possible set of repeated substrings of s, and has an edge from node a to node b if exactly one substring has to be added to the set that corresponds to a in order to obtain the set that corresponds to b. The bottom node corresponds to the empty set while the top corresponds to the set of all repeated substrings of s. The score of a node is defined as the size of a smallest grammar obtained using the optimization of the occurrences replacement with s and the substrings in the node. As an example, the score of the bottom node is the size of grammar $S \to s$ and the score of the top node corresponds to the size of a smallest grammar that has one non-terminal for s and for each repeated substring of s.

There exists a node in the lattice whose score is the size of a smallest grammar. But, this optimal set of substrings cannot be efficiently computed because the lattice that has to be explored is exponentially big. ZZ explores it by an alternation of two different phases: *bottom-up* and *top-down*. The bottom-up can be started at any node, it moves upwards in the lattice and at each step it looks among its immediate descendants for the one with the lowest score. In order to determine which is the one with the lowest score, it inspects them all. It stops when no descendant has a better score than the current one. As in bottom-up, top-down starts at any given node but it moves downwards looking for the node with the smallest score among its immediate ancestors. Going up or going down from the current node is equivalent to adding or removing a substring to or from the set of substrings in the current node respectively.

ZZ starts at the bottom node, that is, the node that corresponds to the grammar $S \to s$ and it finishes when no improvements are made in the score between two bottom-up–top-down iterations.

For example, suppose that there are 5 substrings that occur more than once in α and that they all have length greater than two. Let these strings be numbered from 0 to 4. We start the ZZ algorithm at the bottom node. It inspects nodes $\{0\}$, $\{1\}$, $\{2\}$, $\{3\}$, and $\{4\}$. Suppose that $\{1\}$ produces the best grammar, then ZZ moves to that node and starts over exploring the nodes above it. Figure 2

shows a part of the lattice being explored. Dotted arrows point to nodes that are explored while full arrows points to nodes having the lowest score. Suppose that the algorithm then continues up until it reaches node $\{1, 2, 3\}$ where it can not go up any further. Then ZZ starts the top-down phase, going down to node $\{2, 3\}$ where it can no go any lower. At this point a bottom-up–top-down iteration is done and the algorithm starts over again. It goes up, suppose that it reaches node $\{2, 3, 4\}$ where it stops. Bold circled nodes correspond to nodes were the algorithm switches phases, grey nodes corresponds to nodes with the best score among its siblings.

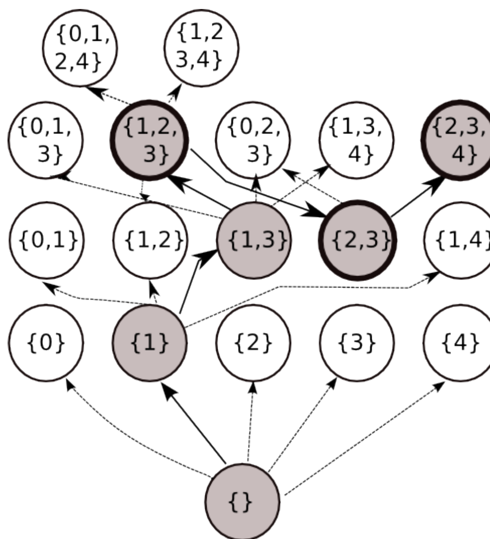

Fig. 2. The fraction of the lattice that is explored by the ZZ algorithm.

Computational Complexity. In the previous section we showed that the computational complexity of computing the score function for each node is $O(n \times m^2)$, where n is the length of the target string and m is the number of substrings in the node. Every time ZZ looks for a substring to add or remove it has to inspect all possible candidates with the aim of finding the one that minimizes the score. Depending on the number of substrings that are already in the node, there might be at most $O(n^2)$ candidate strings. As a consequence, each step upwards or downwards is made in $O(n^2 \times n \times m^2)$. Next, we need to give an upper bound for the length of the path that is potentially traversed by the algorithm. In order to define it, we first note two important properties: the score of the bottom node is equal to n and the score of any node containing more than $n/2$ substrings is at least n. The first one is trivially true, while the second is true because, since every rule body contains at least two symbols, if there were $n/2$ rules, then the grammar size would be at least n. The bottom-up phase visits at most $n/2$ nodes, and consequently, the top-down can only go down at most $n/2$ steps. Adding them together, it turns out that a bottom-up, top-down iteration traverses at most n nodes. Now, each of these iteration decreases the score bt at least 1, otherwise the algorithm stops. Since the initial score is n plus the fact that the score is always positive, it is true that there can be at most n bottom-up–top-down iterations. This results in a complexity for the ZZ algorithm of $O(n^5 \times m^2)$.

4 Experiments

In this section we experimentally compare our algorithms with the classical ones from the literature. For this purpose we use the Canterbury Corpus [20] which is a standard corpus for comparing lossless data compression algorithms. Table 1 lists the sequences of the corpus together with their length and number of repeats of length greater then one.

Table 1. Corpus statistics

sequence	length	# of repeats
alice29.txt	152,089	220,204
asyoulik.txt	125,179	152,695
cp.html	24,603	106,235
fields.c	11,150	56,132
grammar.lsp	3,721	12,780
kennedy.xls	1,029,744	87,427
lcet10.txt	426,754	853,083
plrabn12.txt	481,861	491,533
ptt5	513,216	99,944,933
sum	38,240	666,934
xargs.1	4,227	7,502

Not all existing algorithms are publicly available, they resolve in different way the case when there are more then two words with the best score, they do not report results on a standard corpus or they use different definitions of size of a grammar. In order to standardize the experiments and score, we implemented all the offline algorithms presented in this paper in the IRR framework. For the sake of completeness, we also add to the comparison LZ78 and Sequitur. Note that we have post-processed the output of the LZ78 factorizations to transform them into context-free grammars. The first series of experiments aims at comparing these classical algorithms and are shown in the middle part of Table 2. On this benchmark, we can see that IRR-MC outputs always the smallest grammar, which are in average 4.22% smaller then those of the second best (IRR-MO), confirming the partial results of [9] and showing that IRR-MC is the current state-of-the-art practical algorithm for this problem.

Then we evaluate how the optimization of occurrences improves IRR algorithms. As shown in the IRRCOO column of Table 2, each strategy for choosing the word is improved by introducing the occurrence optimization. The sole exceptions are for the MO strategy on grammar.lsp and xargs.1, but the difference in these cases is very small and the sequences are rather short. More important, we can see that IRCOO-MC is becoming the new state-of-the-art algorithm, proposing for each test a smaller grammar than IRR-MC, and being outperformed only on `plrabn12.txt` by IRCOO-MO.

If we are given more time, these results can still be improved by using ZZ. As shown in column ZZ of Table 2, it obtains in average 3.12% smaller grammars than IRR-MC, a percentage that increases for the sequences containing natural language (for instance, for `alice29.txt` the gain is 8.04%), while it is lower for other sequences (only 0.1% for `kennedy.xls` for example). For the latter case, one can remark that the compression ratio is already very high with IRR-MC and that it may be difficult or impossible to improve it, the last few points of the percentage gain being always the hardest to achieve. As expected, ZZ improves over previous approaches mainly because it explores a much wider fraction of search space. Interestingly enough, the family of IRRCO algorithms also improves state of the art algorithm but still keeps the greedy flavour, and more importantly, it does so with a complexity cost similar to pure greedy approaches. The price to be paid for computing grammars with ZZ is its computational complexity. This problem already showed up with `plrabn12.txt`, `lcet10.txt` (where each

repeat individually does not compress much the sequence, so lots of iterations are necessary) and `ptt5` (which contains about 99 millions of repeats).

It is interesting to know whether the structure of the grammars found by occurrence optimization are simply some refinements of the ones found by classical algorithms or whether they differ completely. For the inference of the structure of the sequence, this is even crucial. This subject deserves a more complete study. We present here the result of comparing the structures returned by IRR-MC and ZZ on the typical test case `asyoulik.txt`. In that case the ZZ grammar is 6.6% smaller than that of IRR-MC, but using the standard unlabeled precision and recall metric [21], gives an F-measure – which is roughly speaking the measure of how similar both structures are – is only 34.6%. Moreover, the F-measure of non-crossing brackets – a measure of how compatible the structures are – is also very low: 35.8%. We can see that the size improvement achieved by the algorithm optimizing the choice of the occurrences has a dramatic effect on the structure found. The same kind of behavior can be already seen between IRR-MC and IRCOO-MC, the F-measure and the F-measure of non-crossing brackets being in that case 55.1% and 56.9% respectively, for a size improvement of 2.9%.

Table 2. Grammar sizes on the Canterbury corpus. The files over which ZZ did not finished are marked with a dash.

	Algorithms from the literature					Optimizing occurrences			
			IRR			IRCOO			
Sequences	Sequitur	LZ78	MO	ML	MC	MO	ML	MC	ZZ
alice29.txt	49,147	116,296	42,453	56,056	**41,000**	39,794	5,235	**39,251**	**37,701**
asyoulik.txt	44,123	102,296	38,507	51,470	**37,474**	36,822	48,133	**36,384**	**35,000**
cp.html	9,835	22,658	8,479	9,612	**8,048**	8,369	9,313	**7,941**	**7,767**
fields.c	4,108	11,056	3,765	3,980	**3,416**	3,713	3,892	**3,373**	**3,311**
grammar.lsp	1,770	4,225	1,615	1,730	**1,473**	1,621	1,704	**1,471**	**1,465**
kennedy.xls	174,585	365,466	167,076	179,753	**166,924**	166,817	179,281	**166,760**	**166,704**
lcet10.txt	112,205	288,250	92,913	130,409	**90,099**	90,493	164,728	**88,561**	–
plrabn12.txt	142,656	338,762	125,366	180,203	**124,198**	**114,959**	164,728	117,326	–
ptt5	55,692	106,456	45,639	56,452	**45,135**	44,192	53,738	**43,958**	–
sum	15,329	35,056	12,965	13,866	**12,207**	12,878	13,695	**12,114**	**12,027**
xargs.1	2,329	5,309	2,137	2,254	**2,006**	2,142	2,237	**1,989**	**1,972**

5 Conclusions

We propose to separate the choice of the words from the choice of the occurrences where they are going to be rewritten in algorithms searching for a smallest grammar. First we improve classical offline algorithms by optimizing at each step the choice of the occurrences. The separation allowing to define the search space as a lattice over sets of repeats, we propose then a new algorithm that explores this search space by adding, but also removing, repeats to the current set of words. Our experiments show that both approaches outperform state-of-the-art algorithms.

The optimization of the choice of occurrences opens new perspectives when searching for the smallest grammars, especially for the inference of the structure of sequences. In future work, we want to study how this scheme helps actually to find better structure on real applications.

References

1. Charikar, M., Lehman, E., Liu, D., Panigrahy, R., Prabhakaran, M., Sahai, A., Shelat, A.: The smallest grammar problem. Information Theory, IEEE Transactions on **51** (2005) 2554–2576
2. Rytter, W.: Application of Lempel-Ziv factorization to the approximation of grammar-based compression. Theoretical Computer Science **302** (2003) 211 – 222
3. Kieffer, J., Yang, E.H.: Grammar-based codes: a new class of universal lossless source codes. IEEE Transactions on Information Theory **46** (2000)
4. Ziv, J., Lempel, A.: Compression of individual sequences via variable-rate coding. IEEE Transactions on Information Theory **24** (1978) 530–536
5. Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. IEEE Transactions on Information Theory **23** (1977) 337–343
6. Nevill-Manning, C.G.: Inferring Sequential Structure. PhD thesis, University of Waikato (1996)
7. Wolff, J.: An algorithm for the segmentation of an artificial language analogue. British Journal of Psychology **66** (1975)
8. Bentley, J., McIlroy, D.: Data compression using long common strings. In: Data Compression Conference. (1999) 287 – 295
9. Nevill-Manning, C., Witten, I.: On-line and off-line heuristics for inferring hierarchies of repetitions in sequences. In: Data Compression Conference, IEEE (2000) 1745–1755
10. Apostolico, A., Lonardi, S.: Off-line compression by greedy textual substitution. Proceedings of the IEEE (2000)
11. Larsson, N., Moffat, A.: Off-line dictionary-based compression. Proceedings of the IEEE **88** (2000) 1722–1732
12. Nakamura, R., Inenaga, S., Bannai, H., Funamoto, T., Takeda, M., Shinohara, A.: Linear-time text compression by longest-first substitution. Algorithms **2** (2009) 1429–1448
13. Lanctot, J.K., Li, M., Yang, E.H.: Estimating DNA sequence entropy. In: ACM-SIAM Symposium on Discrete Algorithms. (2000) 409–418
14. Evans, S.C., Kourtidis, A., Markham, T., Miller, J.: MicroRNA target detection and analysis for genes related to breast cancer using MDLcompress. EURASIP Journal on Bioinformatics and Systems Biology (2007)
15. Nevill-Manning, C.G., Witten, I.H.: Identifying hierarchical structure in sequences: A linear-time algorithm. Journal of Artificial Intelligence Research **7** (1997)
16. Marcken, C.D.: Unsupervised language acquisition. PhD thesis, Massachusetts Institute of Technology (1996)
17. Sakakibara, Y.: Efficient learning of context-free grammars from positive structural examples. Inf. Comput. **97** (1992) 23–60
18. Sakamoto, H., Maruyama, S., Kida, T., Shimozono, S.: A space-saving approximation algorithm for grammar-based compression. IEICE Transactions **92-D** (2009) 158–165
19. Schuegraf, E.J., Heaps, H.S.: A comparison of algorithms for data base compression by use of fragments as language elements. Information Storage and Retrieval **10** (1974) 309–319
20. Arnold, R., Bell, T.: A corpus for the evaluation of lossless compression algorithms. In: Data Compression Conference, Washington, DC, USA, IEEE Computer Society (1997) 201
21. Klein, D.: The Unsupervised Learning of Natural Language Structure. PhD thesis, University of Stanford (2005)