Searching for Smallest Grammars on Large Sequences and Application to DNA☆

Rafael Carrascosa[a], François Coste[b], Matthias Gallé[b,*], Gabriel Infante-Lopez[a,c]

[a] *Grupo de Procesamiento de Lenguaje Natural Universidad Nacional de Córdoba, Argentina*
[b] *Symbiose Project IRISA/INRIA Rennes-Bretagne Atlantique, France*
[c] *Consejo Nacional de Investigaciones Científicas, Argentina*

Abstract

Motivated by the inference of the structure of genomic sequences, we address here the smallest grammar problem. In previous work, we introduced a new perspective on this problem, splitting the task into two different optimization problems: choosing which words will be considered constituents of the final grammar and finding a minimal parsing with these constituents. Here we focus on making these ideas applicable on large sequences. First, we improve the complexity of existing algorithms by using the concept of maximal repeats when choosing which substrings will be the constituents of the grammar. Then, we improve the size of the grammars by cautiously adding a minimal parsing optimization step. Together, these approaches enable us to propose new practical algorithms that return smaller grammars (up to 10%) in approximately the same amount of time than their competitors on a classical set of genomic sequences and on whole genomes of model organisms.

Keywords: linguistics of DNA, smallest grammar problem, structural inference, maximal repeats

1. Introduction

While genome sequencing projects are producing an ever increasing amount of DNA sequences, the challenge in the post-genomic era is now to decipher what has been popularly named "the language of life" [1].

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

*Corresponding author

Email addresses: `rafacarrascosa@gmail.com` (Rafael Carrascosa), `francois.coste@irisa.fr` (François Coste), `matthias.galle@irisa.fr` (Matthias Gallé), `gabriel@famaf.unc.edu.ar` (Gabriel Infante-Lopez)

The linguistic metaphor has been used indeed for a long time in molecular biology, and applying computational linguistics tools to represent and handle biological sequences *in silico* is a natural continuation of this metaphor. As advocated in particular by Searls [2], formal grammars such as the ones introduced by Noam Chomsky [3] to describe natural languages and study syntax acquisition by children, are good candidates for processing sequences in computational biology. The main difficulty in this approach is that, in contrast with all the studies available on natural languages, little is known about the syntax of DNA, as shown for instance by the lack of reliable definitions of "words", "sentences" or "punctuation marks".

As a first step towards better understanding DNA syntax, in this paper we address the problem of automatically discovering the structure of a (long) DNA sequence in a grammatical inference framework. In the lack of background knowledge or any other learning bias, the application of Occam's Razor principle suggests to find a grammar as small as possible that describes the given sequence, assuming that this smallest structure will unveil the eventual hidden structure. This problem can be stated formally as the classical Smallest Grammar Problem [4]: given a sequence, find a context-free grammar of minimal size that generates this and only this sequence. The problem is known to be NP hard, and several algorithms that approximate the solution using heuristics have been proposed. Different approaches offer a different trade-off between the speed of the algorithm and the size of the grammar found. For the discovery of structure, size is important: while a small improvement in the size of the grammar could not worth the effort for compression applications, it can lead to a dramatic change in the structure found. The complexity of the algorithms is still an important issue since algorithms have to be able to handle large DNA sequences such as classical genomes in a reasonable amount of time. Our contribution is both on improving the efficiency of the algorithms and on reducing the size of the grammars returned, by focusing on the selection of the "words" to consider and on the optimization of their occurrences in the text.

The outline of the paper is the following. In Section 2 we analyze previous work and compare existing algorithms on a classical set of genomic sequences of moderate size. In Section 3 we consider the choice of the "words": using maximal repeats instead of repeats allows us to focus on the interesting words and to decrease the number of candidates from a quadratic amount to a linear amount with respect to the size of the sequences. In Section 4 we review our previous results that show how to optimize the choice of the occurrences of the selected words and give three different algorithms using this optimization. Experiments on the classical corpus in Section 5 allow us to compare our approach with state-of-the art algorithms. The results show that our approach allows to find smaller grammars with an average size gain ranging from 4% to 10%, according to the algorithm used. Our faster algorithm, whose running time is comparable to the best identified pre-existing algorithm, can handle bigger sequences: it can run on whole genomes of classical model organisms and is able to return grammars up to 10% smaller than the previous state-of-the-art algorithm on these sequences.

1.1. Definitions and Notation

We begin 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 $j < i$. We extend every string on both sides with a special character \$ that does not appear in the original string, so that $s[0] = s[|s|+1] = \$$. A context-free grammar is a tuple $\langle \Sigma, \mathcal{N}, \mathcal{P}, S \rangle$, where Σ is the set of *terminals* and $\mathcal{N}$ the set of *non-terminals*, $\mathcal{N}$ and Σ disjoint. $S \in \mathcal{N}$ is called the *start symbol* and $\mathcal{P}$ is the set of *productions* (or *rules*). 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 $(\Sigma \cup \mathcal{N})^*$. We say $\alpha \overset{1}{\Rightarrow} \beta$, if α is of the form $\delta C \delta'$, $\beta = \delta \gamma \delta'$ and $C \to \gamma$ is a production. A succession $\alpha \overset{1}{\Rightarrow} \alpha_1 \overset{1}{\Rightarrow} \dots \overset{1}{\Rightarrow} \beta$ is called a *derivation* and in this case we say that α *produces* β and that β *derives* from α (denoted by $\alpha \Rightarrow \beta$).

Given a non-terminal N, its *constituents* ($\text{cons}(N)$) are the possible strings of terminals that can be derived from N. Formally, $\text{cons}(N) = \{w \in \Sigma^* : \omega \Rightarrow w\}$. The constituents of a grammar are all the constituents of its non-terminals. The language is the set of constituents of the axiom S, $\text{cons}(S)$.

Because the Smallest Grammar framework seeks a context-free grammar whose language contains one and only one string, the grammars we consider here neither branch (every terminal occurs at most once in all right-hand sides of rules) nor loop (if B occurs in any derivation starting with A, then A will not occur in a derivation starting with B). This makes such a grammar equivalent to a straight-line program if the grammar is in Chomsky Normal Form [5].

Several definitions of the grammar size exist. Following [6], we define the *size of the grammar* G, noted $|G|$, as the length of the string that results from the concatenation of the right-hand sides of the grammar rules separated by markers:

$$|G| = \sum_{A \to \alpha \in \mathcal{P}} (|\alpha| + 1) \tag{1}$$

This definition has the advantage over others that it corresponds to the number of symbols necessary to represent the grammar in an unambiguous straightforward way. As an example, the grammar:

$S \to aN_2N_2N_1N_1a, \quad N_1 \to abN_2a, \quad N_2 \to bab$

is encoded by the string $aN_2N_2N_1N_1a|abN_2a|bab|$ of size 16.

2. Previous Work

The problem of finding a small grammar for a sequence has gotten attention from different communities and has been addressed with several different algorithms. In this paper, we are particularly interested in “practical” algorithms, this means, algorithms that run in a practical amount of time and that were designed to find small grammars in the general case.

Much work was done to find algorithms that ensure a low asymptotic upper bound of the ratio between the size of the returned grammar and the smallest

size. The best such upper bound is $\mathcal{O}(\log(n/g))$ [4, 7, 8], but these theoretical algorithms do not necessarily guarantee a good behavior in practice. It is known [4] that the size of the LZ77 factorization of a sequence is a lower bound on the size of a smallest grammar for this sequence, which is the reason that the best approximation algorithms are based on this decomposition. We computed the LZ77 factorization on the Canterbury corpus [9], a well-known corpus used to compare general purpose data compressors. For all but one file (namely, *ptt5*) the size of the LZ77 decomposition is bigger than $n/\log_e(n)$, which means that the trivial grammar $\langle \Sigma, \Sigma \cup \{S\}, \{S \rightarrow s\}, S \rangle$ of size $n+1$ is already within an $\log n$ factor of a smallest grammar for all sequences but one of the Canterbury corpus. So, as it is often the case, the constant factor hidden in the Big-O notation can have dramatic consequences in practice.

We review briefly the algorithms we use here. Sequitur [10] is a fast, on-line algorithm which processes the sequences from left to right maintaining two invariants: every digram appears only once in the grammar ("digram uniqueness") and every rule is used at least once ("rule utility"). Supposing constant-time look-up in a hash-table, the algorithm is linear. In [11], Sequitur is modified to guarantee that no two non-terminals produce the same terminal string. This allows to decrease the size of the final grammar, at the cost of breaking the linearity of the algorithm. The new algorithm is named Sequential in [4]. LongestFirst was introduced in 1999 [12] as a general purpose compressor. The algorithm iteratively selects the longest repeat in the sequence, extracts it and replaces all the occurrences of this repeat with a pointer. Later it was modified in order to also take into account the right-hand side of previously introduced rules. This both enriches the grammar and improves its compression capacity. A correct linear version was given recently in [13]. RePair [14] (we refer to this paper for references on earlier similar algorithms) takes a similar approach, but instead of selecting the longest repeat, it selects the most frequent digram. A linear algorithm exists for this, too. Finally, a third option is to select the repeat that greedily compresses best the grammar. This idea was studied in [15] to define Greedy, which was applied in [16] to compress biological sequences. Greedy measures the size of a grammar by counting the number of bits it would take to encode a sentence, and uses an estimation of this measure to select a repeat in each iteration. The same principle, but using the size of the grammar rather than the number of bits, was used in [6] to implement Compressive and compare it with other algorithms. Finally, Bisection [11] divides the sequence recursively in two parts (of equal size if the size of the sequence is a power of two, if not into one part of size of the maximal power of two that is possible, and the rest). Equal subsequences are assigned the same non-terminal

2.1. Comparison on DNA sequences

Among the good candidates to be applied to biological sequences, LongestFirst, RePair and Greedy behave very similarly: they are all off-line, and they iterate over the grammar that is being built, selecting in each iteration a

repeat that maximizes a score function. In order to compare in a uniform framework the behavior of the different score functions, we implemented them in a general schema that we called IRR (for *Iteratively Repeat Replacement*). First, the grammar being built is initialized with a unique initial rule $S \to s$ where s is the input sequence, and then IRR proceeds iteratively. At each iteration, a word ω occurring more than once in the grammar 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 and a new rewriting rule $N \to \omega$ is added to the grammar.

The choice of the score function instantiates different algorithms. As we mentioned, the most popular ones are maximal length (ML), most frequent (MF), and most compressive (MC) which chose the repeat that is longest, most frequent or compresses best the grammar respectively. The three corresponding algorithms are called IRR-ML, IRR-MF, IRR-MC. See [17] for details.

We compare SEQUITUR, SEQUENTIAL, BISECTION, IRR-ML, IRR-MF and IRR-MC on a standard DNA corpus [18][1]. A description of the sequence of this corpus is given in Table 1. For SEQUITUR we downloaded the version from its author website[2] and for BISECTION and SEQUENTIAL we used Y. Ponty's implementation[3]. We used our implementation for the IRR algorithms. Results are reported in Table 2. As it can be seen, IRR-MC reveals again as the algorithm that gets the smallest grammar. The only algorithm that gets close is IRR-MF, beating it once by 0.05%. For all other sequences, IRR-MC gets smaller grammars, with a difference that varies from 0.01 to 25%. The latter result is obtained for the sequence `humghcs`, that is know to contain a high number of repeats. Other sizes are given in percentage with respect to the size of the grammar given by IRR-MC. So for example, SEQUITUR's grammar are on average 7.65% bigger than those found by IRR-MC.

3. Choice of Relevant Constituents

Inside the IRR schema, the different score functions define different alternatives to choose what will become the constituents of the grammar. IRR reduces the possible candidates to the subset of all repeats. This seems logical, as on one hand the final grammar must represent the original sequence in an exact way ("lossless" in terms of compression) and on the other hand the objective of producing small grammars makes non-repeated subwords worthless to consider.

However, to consider all repeats as eventual constituents yields a lot of redundancy: if α is repeated, so will be every subsequence of α. But if this subsequence does not appear elsewhere in the sequence, then it seems pointless to consider it as a constituent. This motivated us to consider the set of

[1] downloaded from `http://web.unipmn.it/~manzini/dnacorpus/historical/`
[2] `http://sequitur.info/`
[3] `http://yann.ponty.free.fr/approximations.html`

Sequence	length	$\frac{\text{\# repeats}}{length}$	description
chmpxx	121,024	0.82	marchantia polymorpha (liverwort) chloroplast
chntxx	155,844	0.77	tobacco chloroplast
hehcmv	229,354	1.46	human cytomegalovirus (strain AD169)
humdyst	38,770	0.77	human dystrophin gene (chr X)
humghcs	66,495	13.77	human growth hormone and chorionic somatomammotropin genes (chr 17)
humhbb	73,308	9.01	human beta globin region (chr 11)
humhdab	58,864	1.21	human contig sequence comprising 3 cosmids (HDAB, HDAC, HDAD)
humprtb	56,737	1.07	human hypoxanthine phosphoribosyltransferase (chr X)
mpomtcg	186,609	1.36	mitochondria of marchantia polymorpha (liverwort)
mtpacga	100,314	0.97	mitochondria of podospora anserina (a filamentous fungus)
vaccg	191,737	2.21	vaccinia virus

Table 1: Description of the DNA corpus we used through this paper. The third column is the result of dividing the number of repeats of the sequence by the length of the sequence.

Sequence	Sequitur	Sequential	Bisection	IRR-ML	IRR-MF	IRR-MC
chmpxx	5.61 %	3.62 %	167.94 %	59.35 %	0.01 %	**28706**
chntxx	5.93 %	2.83 %	174.29 %	58.88 %	0.03 %	**37885**
hehcmv	4.67 %	3.63 %	178.94 %	61.09 %	0.09 %	**53696**
humdyst	5.92 %	3.46 %	160.28 %	53.29 %	0.02 %	**11066**
humghcs	20.3 %	46.36 %	250.92 %	36.32 %	25.46 %	**12933**
humhbb	7.16 %	7.99 %	176.2 %	54.72 %	2.27 %	**18705**
humhdab	9.77 %	6.42 %	169.64 %	51.74 %	0.27 %	**15327**
humprtb	7.74 %	5.47 %	169.96 %	52.94 %	0.35 %	**14890**
mpomtcg	5.62 %	5.08 %	182.07 %	59.01 %	0.9 %	**44178**
mtpacga	6.05 %	4.51 %	169.52 %	57 %	0.29 %	**24555**
vaccg	5.37 %	3.17 %	177.56 %	61.62 %	**-0.05** %	43701
average	7.65 %	8.41 %	179.76 %	55.09 %	2.69 %	–

Table 2: Final grammar size of classical smallest-grammar algorithms on the DNA corpus. Absolute numbers are given for IRR-MC only, the others are given as percentage with respect to IRR-MC. The best for each row is boldfaced.

maximal repeats. Maximal repeats were first introduced in order to have a compact representation of all repeats of a sequence, being their number bounded by $\mathcal{O}(n)$ compared to $\mathcal{O}(n^2)$ for normal repeats [19]. Another interesting property of maximal repeats is that their distribution on genomic sequences follows Zipf's law [20], from which the authors of [20] conclude that they represent good candidates to be considered as words when linguistic approaches are to be used.

Formally:

Definition 1 (Maximal Repeat). *Given the occurrence of a word w at position i of sequence s, we define its* context *as the tuple $\langle s[i-1], s[i+|w|]\rangle$. A word w is a maximal repeat in sequence s if it appears at least two times in sequence s and if it occurs at least two times with different contexts $\langle a, b\rangle$ and $\langle a', b'\rangle$ such that $a \neq a'$ and $b \neq b'$.*

In this section we analyze the behavior of IRR-like algorithms if instead of computing the score function for every repeat, we consider only maximal repeats. For IRR-ML, the chosen word is always a maximal repeat and for IRR-MF, there is always a maximal repeat that has maximal score[4]:

Observation 1.

1. *If $f_{ML}(\omega, \mathcal{P}) = \max\limits_{\alpha \in repeats(\mathcal{P})} f_{ML}(\alpha, \mathcal{P})$ then ω is a maximal repeat.*
2. *There is always a maximal repeat ω s.t. $f_{MF}(\omega, \mathcal{P}) = \max\limits_{\alpha \in repeats(\mathcal{P})} f_{MF}(\alpha, \mathcal{P})$*

For the case of IRR-MC, we characterize the conditions to have a non-maximal repeat with maximal score. Note that, for every non-maximal repeat ω, there is one maximal repeat ω', such that ω' contains strictly ω and both appear the same number of times. Supposing that $|\omega'| = |\omega| + 1$ and that k occurrences of ω' were eliminated to obtain the canonical list of occurrences, then $f_{MC}(\omega, \mathcal{P}) > f_{MC}(\omega', \mathcal{P})$ if and only if $o_{\mathcal{P}}(\omega) - 1 < |\omega| * k$. At the same time, supposing that the distribution over the sequence is i.i.d., the probability that a word ω is a non-maximal repeat is $2 * \left(\frac{1}{|\Sigma \cup \mathcal{N}|}\right)^{(o_{\mathcal{P}}(\omega)-1)}$. Recall that $|\mathcal{N}|$ increases by one in each iteration of IRR. Both equations indicate that in order to find a case where f_{MC} is maximal for a non-maximal repeat, this repeat must have a low number of occurrences. However, in this case f_{MC} would assign it a lower score. So, in practice, such cases should not appear too frequently. Detailed explanation for the given equations can be found in Appendix A.

Our experiments confirmed this: in all instances but one of our test corpus, IRR-MC behaves as the version of IRR-MC that only looks at maximal repeats. In each iteration, both algorithms chose the same repeat and consequently at

[4] The original RePair algorithm considers only digrams. In this case, a non-maximal repeat could be selected.

the end of the execution, both algorithms return the same grammar. File vaccg, where the two algorithms produce different grammars, presents an instance of the situation we described above, but the grammar returned by the algorithm that looks only at maximal repeats is only four symbols bigger than the one returned by IRR-MC.

On top of yielding almost equivalent results in faster time, the use of maximal repeats has the nice property that – under certain mild conditions – the grammar IRR returns is *irreducible* [11], independently of the score function being used. A detailed description and proof for this can be found in Appendix B.

The total number of times a word occurs in a sequence can be easily computed using a suffix tree structure. But the exact computation of the number of non-overlapping occurrences ($o_{\mathcal{P}}(w)$), is more complicated. The problem of computing this number is known as the *String Statistics Problem*. A solution is based on the construction of the *Minimal Augmented Suffix Tree* (MAST) [21] which permits to compute $o_{\mathcal{P}}(w)$ in time $|w|$. The best known algorithm for the construction of a MAST is in $\mathcal{O}(n \log n)$ [22] and it builds in a first phase a suffix tree. So, even reducing the set of candidates to a linear number using maximal repeats, the total running time for a general IRR schema is still $\mathcal{O}(n^2 \log n)$ (the MAST must be created at every iteration), and requires the rather elaborate construction algorithm for a MAST.

We propose a much simpler approach: we ignore overlapping occurrences and instead of $o_{\mathcal{P}}(w)$ we estimate it by the total number of occurrences of w in $\mathcal{P}$. While this score could be very different from the real contraction that could be achieved by replacing this repeat, our experiments (see Table 3) indicate that over the test corpus there is only a small difference between both grammars, and most of the time the version ignoring overlapping occurrences is actually smaller.

An advantage of only computing the non-overlapping occurrences list for the selected repeat is that the resulting IRR schema, using maximal repeats, is of time $\mathcal{O}(n^2)$, for any score whose computation time is constant. This requires only standard techniques (computation of maximal repeats). Special care should be taken so that the chosen repeat has more than one non-overlapping occurrence. If not, adding this production rule would actually increase grammar size. In such cases we take the next best maximal repeat.

Regarding the gain in execution time, the improvement considering only maximal repeats varies depending on the sequence. On the 557 K nucleotides-long sequence of the maize (*zhea mays*) mitochondrion, known for having a large number of repeats, we reached a speed-up of 6.6 times. For this, we used the classical linear algorithm of [23, 24] based on a suffix array to compute all maximal repeats (the same data structure was used for the original IRR-MC to recover all repeats). Combining both improvements gives an accelerated version of IRR-MC. In Table 3 we indicate the time that it took IRR-MC to run on each of the sequences, and the ratio of the accelerated version and the original. Speed-up varies from two (chntxx) to nine (humghcs). Except otherwise stated, from now on we will assume both of these improvements are included in the

Sequence	Size			Time		
	IRR-MC	Accel.	**Gain**	IRR-MC	Accel.	**Ratio**
chmpxx	28706	28754	**-0.17** %	20.61 s	10.02 s	**0.49**
chntxx	37885	38089	**-0.54** %	33.92 s	16.8 s	**0.50**
hehcmv	53696	53545	**0.28** %	65.48 s	32.21 s	**0.49**
humdyst	11066	11201	**-1.22** %	3.99 s	1.73 s	**0.43**
humghcs	12933	12944	**-0.09** %	49.34 s	5.5 s	**0.11**
humhbb	18705	18712	**-0.04** %	19.62 s	5.01 s	**0.26**
humhdab	15327	15311	**0.1** %	9.55 s	3.77 s	**0.40**
humprtb	14890	14907	**-0.12** %	8.45 s	3.42 s	**0.41**
mpomtcg	44178	44178	**0.0** %	55.44 s	24.6 s	**0.44**
mtpacga	24555	24604	**-0.2** %	17.64 s	8.46 s	**0.48**
vaccg	43701	43491	**0.48** %	54.95 s	23.12 s	**0.42**
Average			**-0.13** %			**0.40**

Table 3: Comparision between IRCC-MC and its accelerated version (using maximal repeats and not considering overlapping for score computation). Time is given in seconds.

algorithm.

4. Choice of Occurrences

IRR algorithms have the advantage of being simple and fast, but they all behave greedily: the choice of a constituent together with the occurrences where it is going to be replaced is fixed and never re-considered. In [17] we prove that for certain sequences IRR fails to find a smallest grammar, regardless of the score function used. This general inconvenience is caused by the fact that the IRR framework gives importance to a good choice of constituents, but ignores the choice of which occurrence of these constituents will be replaced and treats this in a straightforward greedy way.

To remedy the flaws of the IRR framework, we previously proposed to separate the choice of constituents from the parsing of the grammar with these constituents. In [17] we state the problem of finding a minimal grammar *given* a fixed set of constituents, a problem we will call the Minimal Grammar Parsing (MGP) Problem. We give a $\mathcal{O}(n^3)$ time algorithm that takes a set of constituents and outputs a minimal grammar that has this set of constituents. This is done by performing an optimal parsing [25] on each constituent. We will denote this algorithm by *mgp*.

We then use this separation to give a search space for the smallest grammar problem. This search space is the lattice $\langle \mathcal{R}(s), \subseteq \rangle$, where $\mathcal{R}(s) = 2^{\text{repeats}(s)}$. Each node of this lattice corresponds to a subset of the repeats of the original sequence, and together with the sequence s, is the constituent set of a possible grammar for s. Using the size of the grammar obtained with the *mgp* algorithm as score for each node, a global minimum of this lattice corresponds to a smallest grammar.

4.1. ZZ

Zig-Zag (ZZ) is an algorithm that traverses this search space in a hill-climbing approach. It explores this space to search for a node whose score is a local minimum. The algorithm starts at the bottom node (the empty set) and at each node inspects all nodes that are formed by adding one constituent to the current node. The one with best (minimal) score becomes the current node. If no node with a better score than the current score exists, a second phase starts that inspects all nodes that are formed by removing one constituent from the current node. These two phases (bottom-up and top-down) are alternated until no score improvement is made between two bottom-up–top-down iterations. Time complexity of ZZ is bounded by $\mathcal{O}(n^7)$. See [17] for details.

As it can be seen in Table 4, ZZ is very powerful, finding grammars 9.19% smaller on average than the state-of-the-art. Running on the test corpus, on some sequences ZZ finished after a few hours but for two sequences we interrupted computation time after four weeks and report only an intermediate result (note that the final grammar is less or equal than the intermediate result). These sequences are marked with a star in Table 4.

4.2. IRRCOOC

The second algorithm we propose tries to take advantage of the relative speed of IRR, while incorporating the concept of minimal grammar parsing. Instead of computing a minimal grammar parsing for all neighbors that have one constituent more than the current node, it selects the next repeat taking into account its length and number of occurrences in the current grammar. In this sense, it proceeds like IRR, except that, after each iteration, it optimizes the choice of occurrences computing the minimal grammar parsing. This optimization can result in rules that are no more used and their elimination would decrease the grammar size. For this, we define:

Definition 2 (Useless Rule). *Given a set of production rules $\mathcal{P}$, a rule $N \rightarrow \omega$ is* useless *if $(o_{\mathcal{P}}(N) - 1) * (|\omega| - 1) < 2$.*

So, given $\mathcal{P}$, we denote by $clean(\mathcal{P})$ the set of rules where each useless rule $N \rightarrow \omega$ was eliminated and the occurrences of N replaced by ω.

Algorithm 1 presents the algorithm IRRCOOC (for IRR with Choice of Occurrence Optimization and Cleanup): it is based on IRR, where a minimal grammar parsing and a cleanup is performed after each iteration. Recall that computing mgp(η) is in $\mathcal{O}(n^3)$ and every execution of line 7 reduces the size of the grammar by at least one. So, the worst-case complexity of IRRCOOC is bounded by $\mathcal{O}(n^4)$. The internal loop (line 5) is similar to the top-down phase of ZZ.

IRRCOOC finds grammar in average almost 2% smaller, needing five times more time, compared to the accelerated version of IRR-MC. (see Table 4). Unfortunately, it does not seem to scale up very well on bigger sequences. In Figure 1 we plot the user time required to execute IRR-MC and IRRCOOC-MC

on successive prefixes of the Escherichia Coli genome. Note that we incorporated into IRRCOOC the modifications described in Section 3, namely limiting the search only to maximal repeats and ignoring overlapping occurrences when computing the score. Both seem to grow as the square of the time (for the case of IRR-MC this can be better appreciated in Figure 2). The constant hidden in the complexity of IRRCOOC-MC however is much bigger than the one of IRR-MC, becoming unfeasible when applied to sequences bigger than the test corpus.

Algorithm 1 Iterative Repeat Replacement with Occurrence Optimization and Cleanup

IRRCOOC(s, f)

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

1: $\eta \leftarrow \{s\}$
2: $\mathcal{P} \leftarrow \mathcal{P}(\eta)$
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: $\quad \eta \leftarrow \eta \cup \{\omega\}$
5: $\quad$ **repeat**
6: $\quad\quad \eta \leftarrow \{cons(N) : N$ non-terminal of G$\}$
7: $\quad\quad \mathcal{P} \leftarrow mgp(\eta)$
8: $\quad\quad \mathcal{P} \leftarrow clean(\mathcal{P})$
9: $\quad$ **until** $\mathcal{P}$ contains no useless rules
10: **end while**
11: **return** $G(\mathcal{P})$

Note that this algorithm differs from the IRRCOO algorithm presented in [17], in that we add here the clean-up phase.

*4.3. IRRMGP**

Analyzing the time used by IRRCOOC in each instruction reveals that the bottleneck lies in the computation of $mgp(\eta)$. The way IRR chooses its constituents is fast and quite direct, while optimizing the occurrences of the constituents is much more expensive. Several compromise choices are possible in order to reduce the number of times this optimization step is performed. Here we propose to do it only at the end of an IRR execution and not in each iteration. This third proposed algorithm can be found in Algorithm 2. It consists of: running IRR, finding a minimal parsing, throwing away useless rules, and repeating this until no further improvement is made. We call this schema IRRMGP* because it can be seen as several applications of IRR completed by a minimal parsing and cleanup. Note that in Algorithm 2 we apply IRR to a set of production rules rather than to a sequence.

Both the execution of IRR and the occurrence optimization step reduces the size of the grammar by at least one. So, IRRMGP* is in $\mathcal{O}(n^4)$ too. However, we measured again the time needed on successive bigger prefixes of the Escherichia

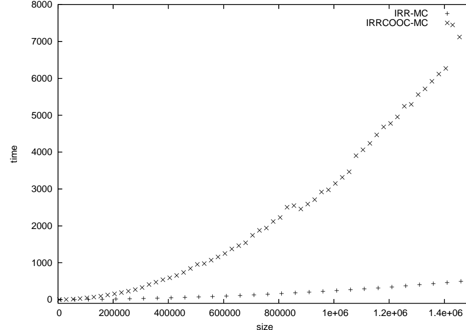

Figure 1: User time for consecutive prefixes of Eschericha Coli for IRR-MC and IRRCOOC-MC. Time is given in seconds and size in bytes

Algorithm 2 IRR plus MGP

IRRMGP$^*(s)$

Require: s is a sequence
1: $\mathcal{P} \leftarrow \{S \rightarrow s\}$
2: **while** $|G| \neq \text{IRR}(\mathcal{P}, f_{MC})$ **do**
3: $\quad \mathcal{P} \leftarrow \text{IRR}(\mathcal{P}, f)$
4: $\quad$ **repeat**
5: $\quad\quad \eta \leftarrow \{cons(N) : N$ non-terminal of G$\}$
6: $\quad\quad \mathcal{P} \leftarrow mgp(\eta)$
7: $\quad\quad \mathcal{P} \leftarrow clean(\mathcal{P})$
8: $\quad$ **until** $\mathcal{P}$ contains no useless rules
9: **end while**
10: **return** $G(\mathcal{P})$

Coli genome. From the result in Figure 2 it can be appreciated that it has the same trend as IRR-MC and takes only slightly more time.

Surprisingly, on the test corpus (Table 4) IRRMGP* outperforms IRRCOOC-MC by obtaining 4.16% smaller grammars on the classical test corpus, taking 27% more time compared to the accelerated version of IRR-MC.

5. More Experiments

Bigger Sequences. We were able to execute IRRMGP* on bigger sequences than those of the standard corpus. We chose model organisms from different kingdoms: *Phage lambda* (virus), *Escherichia coli* (bacteria), *Thalassiosira pseudonana* (chromista protist), *Dictyostelium discoideum* (amoebozoa protist), *Saccaromyces cerevisiae* (fungi), *Ostreococcus tauri* (alga), *Arabidopsis Thaliana* (plant) and *Caenorhabditis elegans* (nematoda). From the two protits (*T.*

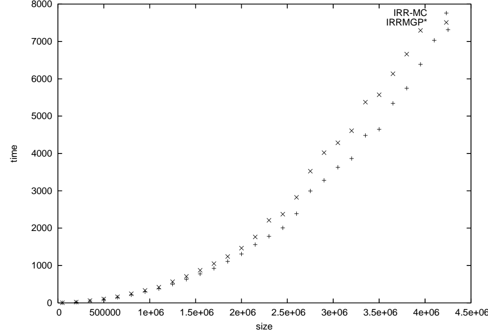

Figure 2: User time for consecutive prefixes of Escherica Coli for IRR-MC and IRRMGP*. Time is given in seconds and size in bytes

sequence	ZZ	IRRCOOC-MC		IRRMGP*	
	size	size	time	size	time
chmpxx	-9.35%	-2.53%	5.62	-4.64%	1.17
chntxx	-10.41%	-2.47%	5.41	-4.74%	1.14
hehcmv	-10.07%[†]	-2.08%	5.31	-5.16%	1.09
humdyst	-8.93%	-2.61%	3.58	-4%	1.19
humghcs	-6.97%	-0.81%	6.07	-2.34%	1.15
humhbb	-8.99%	-1.66%	4.59	-4.43%	1.34
humhdab	-8.7%	-2.07%	4.07	-3.41%	1.12
humprtb	-8.27%	-1.16%	4.39	-3.06%	2.22
mpomtcg	-9.66%	-1.93%	5.53	-3.85%	1.13
mtpacga	-9.64%	-2.41%	4.6	-4.36%	1.2
vaccg	-10.08%[†]	-1.78%	6.36	-5.77%	1.23
average	-9.19%	-1.96%	5.05	-4.16%	1.27

Table 4: Final grammar size of ZZ and IRRCOOC-MC. Size is given as percentage with respect to the final grammar size given by IRR-MC (see Table 2). Fields marked with an [†] are intermediate results. The time for IRRCOOC-MC and IRRMGP* is given as ratio with respect to the time spent by the accelerated version of IRR-MC (see Table 3).

pseudonana and *D. discoideum*) we only took chromosome 1, for *A. Thaliana* we took chromosome 4 and chromosome 3 for *C. Elegans*. For all other cases the sequence corresponds to the whole genome. In each case, the analyzed sequence was the flat DNA sequence, without annotations and where any "N" was deleted. Table 6 shows the results. We report the size of the grammar returned by IRRMGP* and the improvement over IRR-MC. In order to have a relative interpretation we also report the size of the IRRMGP* grammar divided by the length of the sequence. In general, we can see that this number becomes smaller (more redundancy is detected) when the sequence is bigger, but that it is not necessary correlated with the different kingdoms or classification of the analyzed organisms. The average ratio on the classical DNA corpus is 0.23, in the same order as the ratio achieved on the rather small viral genome.

(Dis)similarity. In the preceding section, we saw that including an occurrences' optimization step allows us to find smaller grammars. But maybe more importantly, this improvement in the size of the grammar has big consequences in the structure revealed by the grammar.

Unlabeled precision and recall are the standard measures to compare parse trees in natural language processing (see [26, Section 2.2] for a complete description of these metrics). Basically, the sequence is bracketed according to the parse given by the grammar, and the well-known measures of precision and recall are used to compare the similarity between the sets of brackets. Another useful metric are non-crossing precision and recall, which measures not how similar the structures are, but how compatible (a bracket is incompatible with a structure if it overlaps one of the brackets specified by the structures). In Table 5 we report the F-measure of both metrics applied to the grammars returned by IRR-MC and IRRMGP*. Approximatively half of the brackets are different between both grammars, and the rest is not compatible with the other structure.

6. Perspectives and Conclusions

In this paper, we focused on the smallest grammar problem applied to DNA sequences. On top of their compression capacity, having a (context-free) grammar is appealing for studying DNA because they can give insights on the structure of these sequences. We considered separately the choice of which substrings will become constituents of the grammar and, secondly, which occurrences of these constituents will be replaced by a non-terminal.

This permitted us to present different algorithms that are well suited to generate small grammars and improves the state-of-the-art. The choice of which algorithm to use depends on the size of the original sequence and the desired trade-off between final size and computation time. The algorithms we introduced range from a powerful but computation-expensive one (ZZ), to a much faster (only slightly slower than the state-of-the-art) that permits us to find grammars up to 10% smaller than state-of-the-art when treating whole genomes (with IRRMGP*).

sequence	F_1	NC F_1
chmpxx.chr	48.95	49.00
chntxx.chr	58.30	58.35
hehcmv.chr	56.94	57.00
humdyst.chr	59.45	59.50
humghcs.chr	60.40	60.90
humhbb.chr	53.84	54.05
humhdab.chr	61.30	61.65
humprtb.chr	59.15	59.35
mpomtcg.chr	55.50	55.60
mtpacga.chr	48.65	48.75
vaccg.chr	48.94	48.99
average	55.58	55.74

Table 5: F_1 measure of unlabeled brackets and unlabeled non-crossing brackets between the grammar given by IRR-MC and IRRMGP*

classification	sequence	length	IRRMGP*	$\|G\|/\|s\|$	gain
Virus	P. lambda	48,502	13,061	0.27	-4.25%
Bacterium	E. coli	4,639,675	741,435	0.16	-8.82%
Protist (Chromista)	T. pseudonana chrI	3,031,229	509,203	0.17	-8.15%
Protist (Amoebozoa)	D. discoideum chrI	4,922,989	647,240	0.13	-8.49%
Fungus	S. cerevisiae	12,156,679	1,742,489	0.14	-9.68%
Alga	O. tauri	12,544,522	1,801,936	0.14	-8.78%
Plant	A. Thaliana chrIV	18,582,009	2,561,906	0.14	-9.94%
Nematoda	C. Elegans chrIII	13,783,317	1,897,290	0.14	-9.47%

Table 6: Resulting grammar size for IRRMGP* on some model organism. The last column shows the gain with respect to the size of the grammar of the accelerated version of IRR-MC as percentage.

The performance of ZZ on smaller sequences with respect to IRRMGP* suggests that there is still room for improvement on designing new practical algorithms being more efficient regarding grammar size. Several algorithmical improvements can be made to the IRRMGP* schema, which could be the basis of a wider exploration of the search space. We instrumented the code looking for the components that took most of the computation time. It turns out that the creation of the enhanced suffix array is the most time consuming part and takes up to 90% of the total CPU time. For these experiments, we used the algorithm from [27][5]. A faster suffix array creation algorithm would therefore reduce considerably the total execution time. Moreover, faster algorithms might be obtained if rather than selecting only one constituent in each iteration, one could use all those that do not enter in conflict. This approach can be combined with maintaining dynamically the enhanced suffix array (see [28] for instance).

We considered in this paper the smallest grammar problem in its pure form. Of course the algorithm could be specialized to adapt it to the specificities of DNA by considering and handling simultaneously biological palindromes and both strands of DNA. As DNA sequences presents frequent mutation, it would also be worthwhile to consider inexact patterns. It could even be interesting to sacrifice the lossless property that the original sequence can exactly be regenerated if this could give more insights into a structure behind DNA sequences.

Appendix A. Use of Maximal Repeats in IRR-MC

For IRR-MF and IRR-ML, Observation 1 says that considering only maximal repeats will yield the same grammars as considering all repeats. For the case of MC, however, we do not have an equivalent property. Actually, it could happen that a repeat that maximizes f_{MC} is not maximal. Consider for instance the case of a non-maximal repeat w with two occurrences, both of them with context $\langle a, a \rangle$. If both occurrences of awa overlap (because they occur at position i and $i + |w| + 1$ for some i), then awa would not be considered and the best repeat becomes w. Here we will characterize the condition when a non-maximal repeat maximizes f_{MC}.

If ω is a repeat, then there is exactly one maximal repeat that contains ω and appears the same number of times. We call this maximal repeat $mr(\omega)$. We are interested in non-maximal repeats ω such that $f_{MC}(\omega, \mathcal{P}) > f_{MC}(mr(\omega), \mathcal{P})$. Note that $o_{\mathcal{P}}(\omega) = o_{\mathcal{P}}(mr(\omega)) + k_1$ and $|mr(\omega)| = |\omega| + k_2$ for some positive k_1, k_2, this is, $mr(\omega)$ is k_2 symbols longer the ω and have k_1 occurrences that must be eliminated to have a maximal non-overlapping list. Replacing in the definition of f_{MC}:

$$\begin{array}{rcl} (o_{\mathcal{P}}(\omega) - 1) * (|\omega| - 1) & > & (o_{\mathcal{P}}(\omega) - k_1 - 1) * (|\omega| + k_2 - 1) \\ \equiv \quad \frac{k_1}{k_2} & > & \frac{o_{\mathcal{P}}(\omega) - 1}{|\omega| + k_2 - 1|} \end{array}$$

Supposing that $k_2 = 1$, this gives

[5]implementation downloaded from `http://sites.google.com/site/yuta256/sais`

$$|\omega| * k_1 > o_{\mathcal{P}}(\omega) - 1 \tag{A.1}$$

which is the formula used in Section 3.

At the same time, note that the probability of a repeat to be non-maximal decreases with its number of occurrences and the size of the alphabet. In order to be a non-maximal repeat, a repeat must have all its left-context equal, and all its right-context equal. Supposing that the sequence is i.i.d, we have

$$P(\omega \text{ is not maximal}) = 2 * \left(\frac{1}{|\Sigma|}\right)^{(o_{\mathcal{P}}(\omega)-1)} \tag{A.2}$$

Appendix B. Irreducibility of IRR

In [11] Kieffer and Yang define *irreducibility* for a grammar, and demonstrate that irreducible grammar based codes are universal. We will analyze the conditions when IRR algorithms generates irreducible grammars, independently of the score function used.

Definition 3 (Irreducibility, Kieffer and Yang). *A context-free grammar* $G = \langle \Sigma, \mathcal{N}, \mathcal{P}, S \rangle$ *is said to be irreducible if:*

1. *cons*(S) *is non empty, all rules have a non-empty right-hand sides and each symbol is used at least once in a possible derivation of constituents of cons*(S)
2. *for each non-terminal* A *there is at most one production whose left member is* A
3. *each non-terminal, except* S*, appears at least twice in the right hand members of* $\mathcal{P}$
4. *for* $A, B \in \mathcal{N}, A \neq B\ : cons(A) \neq cons(B)$
5. *no* $\alpha \in (\Sigma \cup \mathcal{N})^*$ *with* $|\alpha| \geq 2$ *appears more than once in non-overlapping positions of the right members of* $\mathcal{P}$

Condition 1 and 2 are trivially true for IRR algorithms, but condition 3 may be violated by the IRR schema if it stops when no further improvement can be made. Nevertheless, it is enough to change the condition of the while loop in order to continue until $\mathcal{P}$ contains no repeats.

Condition 4 is harder to see. A clean demonstration is given in [4, Lemma 6 and 7]. While there the notion of *global algorithm* is different from IRR, their demonstration in these lemmas can be applied without modification to IRR.

Finally, an IRR algorithm may still violate Condition 5. Suppose for example that a non-maximal repeat α is chosen and replaced by N, and that every occurrence of α has as right context of a. If in a future iteration the repeat Na is chosen, then N would occur only once in the grammar. In [4] a special kind of repeat is defined to avoid these cases. Instead of this, the use of maximal repeat gives a more general solution: if it is ensured that the selected word has at least two occurrences in his canonical list with different context, then the resulting grammar is irreducible.

References

[1] G. W. Beadle, M. Beadle, The language of life: an introduction to the science of genetics, American Institute of Biological Sciences, 1966.

[2] D. B. Searls, The language of genes, Nature (2002) 7.

[3] N. Chomsky, Syntactic Structures, Mouton & Co., 1957.

[4] M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, a. shelat, The smallest grammar problem, IEEE Transactions on Information Theory 51 (7) (2005) 2554–2576.

[5] M. Karpinski, W. Rytter, A. Shinohara, An efficient pattern-matching algorithm for strings with short descriptions, Nordic Journal of Computating 4 (1997) 172–186.

[6] C. Nevill-Manning, I. Witten, On-line and off-line heuristics for inferring hierarchies of repetitions in sequences, in: Data Compression Conference, IEEE, 2000, pp. 1745–1755.

[7] W. Rytter, Application of Lempel-Ziv factorization to the approximation of grammar-based compression, Theoretical Computer Science 302 (1-3) (2003) 211 – 222.

[8] H. Sakamoto, S. Maruyama, T. Kida, S. Shimozono, A space-saving approximation algorithm for grammar-based compression, IEICE Transactions 92-D (2) (2009) 158–165.

[9] R. Arnold, T. Bell, A corpus for the evaluation of lossless compression algorithms, in: Data Compression Conference, IEEE Computer Society, Washington, DC, USA, 1997, p. 201.

[10] C. G. Nevill-Manning, I. H. Witten, Compression and explanation using hierarchical grammars, The Computer Journal 40 (2,3) (1997) 103–116.

[11] J. Kieffer, E.-H. Yang, Grammar-based codes: a new class of universal lossless source codes, IEEE Transactions on Information Theory 46 (46) (2000) 737 – 754.

[12] J. Bentley, D. McIlroy, Data compression using long common strings, in: Data Compression Conference, IEEE, 1999, pp. 287 – 295.

[13] R. Nakamura, S. Inenaga, H. Bannai, T. Funamoto, M. Takeda, A. Shinohara, Linear-time text compression by longest-first substitution, Algorithms 2 (4) (2009) 1429–1448.

[14] N. Larsson, A. Moffat, Off-line dictionary-based compression, Proceedings of the IEEE 88 (11) (2000) 1722–1732.

[15] A. Apostolico, S. Lonardi, Off-line compression by greedy textual substitution, Proceedings of the IEEE 88 (2000) 1733–1744.

[16] A. Apostolico, S. Lonardi, Compression of biological sequences by greedy off-line textual substitution, in: Data Compression Conference, IEEE, 2000, pp. 143–153.

[17] R. Carrascosa, F. Coste, M. Gallé, G. Infante-Lopez, Choosing word occurrences for the smallest grammar problem, in: Language and Automata Theory and Applications, 2010, p. 12.

[18] S. Grumbach, F. Tahi, A new challenge for compression algorithms: Genetic sequences, Information Processing and Managment 30 (6) (1994) 875–886.

[19] D. Gusfield, Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology, Cambridge University Press, 1997.

[20] J. Wang, H.-C. Liu, J. J. Tsai, K.-L. Ng, Scaling behavior of maximal repeat distributions in genomic sequences, International Journal of Cognitive Informatics and Natural Intelligence 2 (3) (2008) 31–42.

[21] A. Apostolico, F. Preparata, Data structures and algorithms for the string statistics problem, Algorithmica 15 (5) (1996) 481–494.

[22] G. S. Brodal, R. Lyngsø, A. Östlin, C. N. S. Pedersen, Solving the string statistics problem in time O(n log n), in: International Colloquium on Automata, Languages, and Programming, 2002, pp. 728–739.

[23] M. I. Abouelhoda, S. Kurtz, E. Ohlebusch, Replacing suffix trees with enhanced suffix arrays, J. of Discrete Algorithms 2 (2004) 53 – 86.

[24] S. J. Puglisi, W. F. Smyth, M. Yusufu, Fast optimal algorithms for computing all the repeats in a string, Prague Stringology Club (2008) 161–169.

[25] T. Bell, J. Cleary, I. H. Witten, Text Compression, Prentice Hall, 1990.

[26] D. Klein, The unsupervised learning of natural language structure, Ph.D. thesis, University of Stanford (2005).

[27] G. Nong, S. Zhang, W. H. Chan, Two efficient algorithms for linear suffix array construction, http://www.cs.sysu.edu.cn/nong/ (May 2008).

[28] M. Gallé, P. Peterlongo, F. Coste, In-place update of suffix array while recoding words, International Journal of Foundations of Computer Science 20 (6) (2009) 1025–1045.