Computing Abelian Complexity of Binary Uniform Morphic Words

F. Blanchet-Sadri[a], Daniel Seita[b], David Wise[c]

[a] *Department of Computer Science, University of North Carolina, P.O. Box 26170, Greensboro, NC 27402–6170, USA*
[b] *Computer Science Division, University of California, Berkeley, 387 Soda Hall, Berkeley, CA 94720–1776, USA*
[c] *Department of Mathematical Sciences, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15289, USA*

Abstract

Although a lot of research has been done on the factor complexity (also called subword complexity) of morphic words obtained as fixed points of iterated morphisms, there has been little development in exploring algorithms that can efficiently compute their abelian complexity. The factor complexity counts the number of factors of a given length n, while the abelian complexity counts that number up to letter permutation. We propose and analyze a simple $\mathcal{O}(n)$ algorithm for quickly computing the exact abelian complexities for *all* indices from 1 up to n, when considering binary uniform morphisms. Using our algorithm we also analyze the structure in the abelian complexity for that class of morphisms. Our main result implies, in particular, that the infinite word over the alphabet $\{-1, 0, 1\}$ constructed from the consecutive forward differences of the abelian complexity of a fixed point of a binary uniform morphism is in fact an automatic sequence with the same morphic length. Since the proof produces morphisms that typically contain many redundant letters, we present an efficient algorithm to eliminate them in order to simplify the morphisms and to see the patterns produced more clearly.

Keywords: Formal languages, Algorithms, Analysis of algorithms, Abelian complexity, Morphisms, Morphic words, Automatic sequences

Email addresses: blanchet@uncg.edu (F. Blanchet-Sadri), seita@berkeley.edu (Daniel Seita), dwise@andrew.cmu.edu (David Wise)

1. Introduction

The concept of *abelian complexity* is relatively new as compared to its classical counterpart of *factor complexity*. On the one hand, the factor complexity of a given infinite word w is a function $\rho_w : \mathbb{N} \to \mathbb{N}$ that maps an integer n to the number of distinct factors of w of length n. On the other hand, the abelian complexity of w is a function $\rho_w^{ab} : \mathbb{N} \to \mathbb{N}$ that maps an integer n to the number of distinct *Parikh vectors* of factors of w of length n. Parikh vectors, which appear in the literature under various names such as compomers [5, 6], jumbled patterns [8, 7], permutation patterns [9, 18, 24], commutative closures [19], content vectors [19], to name a few, are vectors that record the frequency of each letter in the factors. So the abelian complexity counts the number of distinct factors of a given length up to letter permutation. For a survey on abelian concepts such as abelian complexity as well as applications, we refer the reader to [11].

We focus on the abelian complexity of those words obtained as fixed points of iterated morphisms starting at some letter. For example, the *Thue-Morse word* is the fixed point starting at 0 of the uniform morphism $0 \mapsto 01, 1 \mapsto 10$ over the binary alphabet $\{0, 1\}$, i.e.,

$$0110100110010110 \cdots .$$

A lot of research has been done on the factor complexity of fixed points of morphisms, e.g., Frid [20] obtained an explicit formula for the factor complexity of the words obtained from a binary uniform morphism.

As to research done on the abelian complexity of fixed points of morphisms, there are two main approaches that have been taken so far. The first approach consists in deriving a formula. This is in general very difficult to do, so it has only been done for a few special cases over a binary or a ternary alphabet: Sturmian words [12], the Thue-Morse word[27], the Tribonacci word, i.e., the fixed point starting at 0 of the morphism $0 \mapsto 01, 1 \mapsto 02, 2 \mapsto 0$ [26], the paper-folding word [22], and quadratic Parry words, i.e., a morphism of the type $0 \mapsto 0^p1, 1 \mapsto 0^q$ with $p \geq q \geq 1$ or of the type $0 \mapsto 0^p1, 1 \mapsto 0^q1$ with $p > q \geq 1$, [2]. Constant and ultimately constant abelian complexity of infinite words has also been studied [13, 28]. The second approach consists in sliding a window of size n on a sufficiently long prefix of a fixed point w of a morphism to count the number of distinct Parikh vectors. This is also

in general very difficult to do as the required prefix's length grows to infinity as n goes to infinity, exhausting computer memory. Reference [30] combines these two approaches: it considers an infinite word w belonging to a subclass of Parry words, and instead of sliding a window of size n on a sufficiently long prefix of w, it finds a walk on a transition diagram of a discrete finite-state automaton constructed from w, a finite graph that is independent of n. Equivalently, it is shown how to find a transition function and an output function that allow the evaluation of the value $\rho_w^{ab}(n)$ in $\mathcal{O}(\log n)$ steps.

Both approaches being difficult, there has also been work on the *asymptotic* abelian complexity of some morphic words, e.g., the fixed point starting at 0 of the non-uniform morphism $0 \mapsto 012, 1 \mapsto 02, 2 \mapsto 1$ [3]. Such works are mainly concerned with the asymptotic behaviors of the abelian complexities rather than their specific values. The classification of the asymptotic growths of the abelian complexities of fixed points of binary morphisms has been undertaken and a complete classification in the case of binary uniform morphisms has been derived [4]. This classification had previously been done for the factor complexity of fixed points of morphisms [14, 15, 16, 17, 23].

There has been little development in exploring algorithms that can efficiently compute abelian complexity values. Our paper presents an efficient algorithm for binary uniform morphisms, and also exposes an interesting mathematical connection between a class of morphisms and the abelian complexities of their fixed points.

The contents of our paper are as follows: In Section 2, we discuss terminology related to the abelian complexity of fixed points of morphisms. In Section 3, we describe and analyze a simple $\mathcal{O}(n)$ algorithm for computing the abelian complexities $\rho^{ab}(n)$ for *all* indices from 1 up to n, when considering binary uniform morphisms. In Section 4, we discuss morphic words and prove some of their properties not shared by fixed points of morphisms. They are constructed from two morphisms σ and τ. In Section 5, using our algorithm, we analyze the structure in the abelian complexity of fixed points of binary uniform morphisms. We state and prove our main result which implies, in particular, that if φ is a binary uniform morphism with morphic length ℓ and the number of zeroes in $\varphi(0)$ minus the number of zeroes in $\varphi(1)$ is not equal to 1, then the infinite word over the alphabet $\{-1, 0, 1\}$ constructed from the forward differences of $\rho^{ab}_{\varphi^\omega(0)}$, where $\varphi^\omega(0)$ denotes the fixed point of φ starting at 0, is an ℓ-automatic sequence. In Section 6, we prove some results on self-similarity. In Section 7, we give an efficient algorithm to eliminate redundancies in automatic sequences. This algorithm is

of interest because the morphisms σ and τ constructed in the proof of our main result typically contain many redundant letters, so it is nice to be able to simplify the morphisms to see the pattern produced more clearly. Finally in Section 8, we conclude with some remarks and open problems for future work.

2. Preliminaries

Let Σ denote an arbitrary alphabet. A *word* w over Σ is a (finite or infinite) sequence of characters from Σ. The character at position i of w is denoted by $w[i]$ (position labelling starts at 0) and the *factor* from position i to position j inclusive by $w[i..j]$. If j is non-inclusive, the factor is denoted by $w[i..j)$. Here $[i..j]$ (resp., $[i..j)$) denotes the set $\{i, i+1, \ldots, j\}$ (resp., $\{i, i+1, \ldots, j-1\}$). When w is finite, the number of characters in w is denoted by $|w|$ and is called the *length* of w. The *empty word* ε is the word of length 0 and Σ^* denotes the set of all finite words over Σ, including ε. Equipped with the concatenation of words, Σ^* forms a monoid with ε acting as the identity.

Define the *factor complexity* ρ_w of a word w over Σ to be the function that, for each positive integer n, returns the number of distinct factors of length n of w. Likewise, define the *abelian complexity* ρ_w^{ab} of a word w over Σ to be the function that, for each positive integer n, returns the number of abelian equivalence classes of factors of length n of w. Here two factors are *abelian equivalent* if one can be obtained from the other by a permutation of characters. If w is understood, we often drop it and simply write ρ, ρ^{ab} respectively. To compute $\rho^{ab}(n)$, we count the number of *Parikh vectors* for all length-n factors, where the Parikh vector of a factor v is the vector whose ith entry is the number of occurrences of the ith letter in v; e.g., the Parikh vector of *ababbca* is $(3, 3, 1)$.

A *morphism* φ over $\Sigma = \{0, 1, \ldots, k-1\}$ is a function $\Sigma^* \to \Sigma^*$ such that if $u, v \in \Sigma^*$, we have $\varphi(uv) = \varphi(u)\varphi(v)$. We denote it as $\varphi = (x_0, x_1, \ldots, x_{k-1})$, where $\varphi(i) = x_i$. The *fixed point* $\varphi^\omega(a)$ of morphism φ over Σ starting at $a \in \Sigma$ is the limit as $n \to \infty$ of $\varphi^n(a)$, if it exists. A morphism is *binary* if it is over a 2-letter alphabet. A morphism φ over Σ is *uniform* if $|\varphi(a)| = |\varphi(b)|$ for all $a, b \in \Sigma$ (it is ℓ-*uniform* if $|\varphi(a)| = \ell$ for all $a \in \Sigma$).

We focus on morphisms φ over the alphabet $\{0, 1\}$ that are non-erasing (i.e., nothing maps to ε), are such that $\varphi(0)[0] = 0$, and have both zeroes and ones in their existing fixed point starting *at zero*. We borrow some

terminology from [4]. Let u be a factor of $\varphi^\omega(0)$. We denote by $z(u)$ the number of zeroes in u. We use z_0 and z_1 as shorthand for $z(\varphi(0))$ and $z(\varphi(1))$, respectively. The functions $z_M(n)$ and $z_m(n)$ return the maximum and minimum number of zeroes of any length-n factor of $\varphi^\omega(0)$. The *morphic length* of φ, denoted ℓ, is equal to $|\varphi(0)|$ (which is equal to $|\varphi(a)|$ for all other letters a if uniformity holds). The *difference* of φ, denoted d, is $|z_0 - z_1|$. The *delta* of φ, denoted Δ, is $z_M(\ell) - \max\{z_0, z_1\}$. For example, let $\varphi = (0001, 0110)$. We clearly have $\ell = 4$ and $d = 1$. Furthermore, $\max\{z_0, z_1\} = 3$ and $\varphi(10) = 011\underline{0000}1$ has a factor of length ℓ with four zeroes, so $\Delta = 4-3 = 1$.

Lemma 1 ([4])**.** *For a binary morphism φ, we have $\rho^{ab}(n) := \rho^{ab}_{\varphi^\omega(0)}(n) = z_M(n) - z_m(n) + 1$.*

We use the standard notation Δf to denote the finite forward difference of f, which is given by $\Delta f(n) = f(n+1) - f(n)$. For any binary word w, we have $|\Delta\rho^{ab}_w(n)| \leq 1$ [4].

A *block* of a factor of the fixed point $\varphi^\omega(0)$ is a maximal group of characters formed from the image of one earlier character under φ. For example, let $\varphi = (001, 110)$. We have the word $w_0 = 0010011101$ as a factor of $\varphi^\omega(0)$. If we underline the individual blocks, we get $\underline{001}\,\underline{001}\,\underline{110}\,\underline{1}$, so w_0 is itself a factor of $\varphi(0011)$, the image under φ of the length-4 factor 0011 of $\varphi^\omega(0)$. Note that depending on φ and our given factor v, there may be multiple ways to divide v into blocks. For example, if $\varphi = (01, 10)$, then $\varphi^\omega(0) = 01101001\cdots$, so the factor 01 occurs as its own block (e.g., the instance starting at position 0) or it can be split into two blocks (e.g., the instance starting at position 3).

A factor w of $\varphi^\omega(0)$ is *generated* by a factor u of $\varphi^\omega(0)$ if w is a factor of $\varphi(u)$, e.g., if $\varphi = (0101, 1110)$, then $w = 011110111$ is generated by $u = 011$. Typically we are only interested in generators u of w such that there exists a copy of w in $\varphi(u)$ that intersects the factors of $\varphi(u)$ coming from the leftmost and rightmost letters of u. Otherwise these letters do not contribute to forming w. From now on, when we talk about a factor u that generates a factor w it is assumed that u satisfies this condition. Related to the leftmost and rightmost letters of a factor of $\varphi^\omega(0)$, the *border* of that factor is the set of characters that form the starting and ending blocks. If there is only one block present, then the border consists of the entire word. Returning to our example above, the word w_0 has a border of length 4.

3. Linear-Time Algorithm for Binary Uniform Morphisms

To compute $\rho^{ab}(n)$ for the fixed point of a *binary* morphism φ starting at 0, we just need $z_m(n)$, the minimum number of zeroes in length-n factors and $z_M(n)$, the maximum number of zeroes in length-n factors (see Lemma 1). Let z_0 be the number of zeroes in $\varphi(0)$ and z_1 the number of zeroes in $\varphi(1)$. We can assume that $z_0 \geq z_1$. If this does not hold, then $\varphi(\varphi(0))$ has more zeroes than $\varphi(\varphi(1))$, so φ^2 has the desired property. Thus, if φ does not satisfy $z_0 \geq z_1$, then we just consider φ^2 instead because φ^2 has the same fixed point as φ.

We describe a fast algorithm to compute $z_m(n)$ for a binary uniform morphism φ with morphic length ℓ. To minimize the number of zeroes in a factor of length n, we want to minimize the number of zeroes in the factor *generating* it. The assumption that $z_0 \geq z_1$ gives the rough idea that we would like to find a factor v of length $\frac{n}{\ell}$ of the fixed point that contains the minimum number of zeroes, and then take $w = \varphi(v)$ to be our optimal factor of length n. However, this does not work perfectly because n may not be a multiple of ℓ, and even if it is, the optimal factor may not be aligned along multiples of ℓ in the fixed point (and therefore may not be the image of some v under φ).

The main idea of Algorithm 1 lies on the remark that, if φ is a morphism with morphic length ℓ, then any word of length $n > \ell$ can be decomposed as $s\varphi(u)p$ where s is a suffix of $\varphi(a)$ for some letter a, p is a prefix of $\varphi(b)$ for some letter b, and the length of aub is smaller than n.

Therefore, our algorithm records four lists, each with a specified starting and ending letter $AB \in \{00, 01, 10, 11\}$. For each list, the row corresponding to n (or just row n for short), $n \geq 2$, contains the minimum number of zeroes in any length-n factor of $\varphi^\omega(0)$ that starts with A and ends with B, if such a factor exists. To compute $z_m(n)$, we just take the minimum of the (at most) four values in row n. This algorithm can easily be modified to compute $z_M(n)$, making it an efficient and fairly straightforward way to compute $\rho^{ab}(n)$.

Algorithm 1. *Given as input a binary uniform morphism φ with morphic length ℓ having a fixed point starting at 0, $\varphi^\omega(0)$, generate the four lists for n values as follows:*

- *Find all factors of length two in $\varphi^\omega(0)$. To do this, find all factors of length two in $\varphi(0)$ and $\varphi(1)$ (this assumes $\varphi(0)$ contains at least one 1;*

if not then the fixed point is just $00\cdots$ *and* $z_m(n) = n$ *for all* n*). Note that if a word of length two,* c_1c_2*, has been found, then after applying* φ *a possibly new word of length two,* $\varphi(c_1)[\ell-1]\varphi(c_2)[0]$*, can be found. Thus for every new word of length two that is found, repeat this process until all words of length two are found or until none of the new words found at some stage produce any undiscovered words.*

- *For each border length* $L \in [2..2\ell]$ *and every letters* $A, B, a, b \in \{0, 1\}$*, construct a border table* AB*, corresponding to the list* AB*, whose entry in row* L *and column* ab *stores the minimum number of zeroes in* w_1w_2 *for any non-empty words* w_1, w_2 *subject to the conditions that* $|w_1| + |w_2| = L$*,* w_1 *is a suffix of* $\varphi(a)$*,* w_2 *is a prefix of* $\varphi(b)$*,* w_1 *starts with* A*, and* w_2 *ends with* B*, if such words exist, or stores a blank otherwise.*

- *For* $n \in [2..\ell]$*, compute row* n *of the four lists by using brute-force over all factors of length two. For each word* c_1c_2 *of length two, search* $\varphi(c_1c_2)$ *for all words of length* n *starting and ending with the specified letters, and record the minimum value found. Every factor* w *of* $\varphi^\omega(0)$ *of length at most* ℓ *is a factor of* $\varphi(v)$ *for some word* v *of length two (since if* w *intersects the images of three distinct letters, then* $|w| \geq \ell + 2$*).*

- *For* $n > \ell$*, compute row* n *of the four lists by using the border tables along with recursion.*

  - *To find a factor* w *of length* n *of* $\varphi^\omega(0)$ *starting with* A*, ending with* B*, and containing as few zeroes as possible, case over all lengths* $L \in [2..2\ell]$*, with* $L \equiv n \pmod{\ell}$*, and all letters* a, b*, to find the minimum number of zeroes in a word* $v = aub$ *of length* $m = \frac{n-L}{\ell} + 2$ *such that* w *is a factor of* $\varphi(v)$*.*
  - *To do this, for fixed* L *and* ab*, break* w *into* m *blocks, such that the* i*th block comes from the image of the* i*th letter of* v*. Let* w_1 *be the first block of* w *and* w_2 *be the last, i.e.,* w_1 *and* w_2 *are the border blocks of* w *satisfying* $|w_1|+|w_2| = L$*,* w_1 *is a suffix of* $\varphi(a)$*,* w_2 *is a prefix of* $\varphi(b)$*,* w_1 *starts with* A*, and* w_2 *ends with* B*. Compute the number of zeroes in* w *by adding the number of zeroes in the border blocks of* w *(which comes from the precomputed value in row* L *of border table* AB*) with the number of zeroes in the interior blocks of* w *which is the number of zeroes in* $\varphi(u)$ *(the number of zeroes in* $v = aub$ *comes from the precomputed value in row* m *of list* ab*).*

– *Calculate the value in row n, list AB as the minimum number of zeroes found among all such w.*

For example, let $\varphi = (001, 110)$, where $\varphi^\omega(0) = 001001110001001110\cdots$. The four border tables, which contain the numbers of zeroes in the optimal border blocks, are

	border table 00				border table 01				border table 10				border table 11			
L	00	01	10	11	00	01	10	11	00	01	10	11	00	01	10	11
2			2					1	1					0		
3	2		3			1		1	2		2			0		1
4	3			2		1	3			1	2		2			1
5	4	2			3	2					3	2			3	1
6		3			4							2			3	

For instance, in border table 00, row 2 is only non-blank in column 10, since a length-2 border means we take the last element of the first block and the first element of the last block, yielding a word of the form $\varphi(1)[2]u\varphi(0)[0]$ where u is some factor of $\varphi^\omega(0)$. This turns out to be $0u0$, a word that starts and ends in a 0, as required.

The following records the first few entries of the four lists:

	list	list	list	list
n	00	01	10	11
2	2	1	1	0
3	2	1	1	0
4	2	1	1	1
5	2	2	2	1
6	3	2	2	1
7	3	2	2	2
8	3	3	3	2
9	4	3	3	2

If we consider $n = 3$, the word 111 is a length-3 factor of $\varphi^\omega(0)$ starting *and* ending in a 1 that contains 0 zeroes, so row 3 of list 11 has a 0. It is simple to compute rows 2 and 3 of these four lists by brute force; for each list the minimum number of zeroes for that list is achieved. After this we can begin the recursive process. It is more instructive to examine a larger row, so assume we have computed rows 2 up to 9. Now, suppose we wish to compute

row 10, list 00. Here, $n = 10$ and $AB = 00$. The only value congruent to 10 modulo 3 in the range [2..6] is 4, so $L = 4$. This gives $m = \frac{n-L}{\ell} + 2 = 4$ for the length of v, which is the word that the optimal word w of length 10 is generated from by applying φ and then taking some factor. So we are looking recursively at the entries in row 4 of our lists. Now we have to case on ab, which are the starting and ending letters of v.

For $ab \in \{01, 10\}$, looking up the entries for row 4 in border table 00 gives blanks, so these values of ab are not possible. For $ab = 00$, we look up row 4 of column 00 in border table 00 and see that the minimal number of zeroes is 3. In this case v has length 4 and starts and ends with 0, so we look up row 4 of list 00 and get the value 2. However, we are only interested in the interior letters of v, since we have precomputed the number of zeroes in the border blocks, so we subtract 2, because the 2 border letters of v are zeroes. Therefore, $v = 0110$. Since $z_1 = 1$, this means that the interior blocks of w contain 2 zeroes, so the optimal word w in this case has $3 + 2 = 5$ zeroes. Finally for $ab = 11$, we can also show that w contains 5 zeroes. Thus, the value for row 10, list 00 is 5.

Theorem 2. *Let φ be a binary uniform morphism. Using Algorithm 1, the values $\rho^{ab}(1), \rho^{ab}(2), \ldots, \rho^{ab}(n)$ can be computed in $\mathcal{O}(n)$ time.*

Proof. There are four possible restrictions on the leftmost and rightmost letters AB of our length-n factor w, four possible restrictions on the letters ab *that generated* the leftmost and rightmost blocks w_1 and w_2 of w, and $2\ell - 1$ possible values for the total border length $L = |w_1| + |w_2|$, where ℓ is the morphic length.

To determine the minimum possible number of zeroes in w given a restriction on the leftmost and rightmost letters AB, we consider $L \equiv n \pmod{\ell}$ to determine which border length values L in the range $[2..2\ell]$ are possible.

For each of the four possible choices 00, 01, 10, or 11 for letters ab, we look up the way to arrange w_1 and w_2 to minimize the total number of zeroes, and add in the total number of zeroes generated by the optimal factor u of length $m - 2 = \frac{n-L}{\ell}$ of the factor $v = aub$ such that w is a factor of $\varphi(v)$. The number of zeroes in w_1w_2 can be calculated from row L and column ab of border table AB, while the number of zeroes in v can be calculated from row m of list ab.

Assuming integer computation can be done in constant time, this process takes constant time, so repeating this process for all values up to a user-inputted value n means we can compute the first n rows of the four lists for

z_m in $\Theta(n)$ time. In combination with the z_M computation, it takes $\mathcal{O}(n)$ time to compute all the values $\rho^{ab}(1), \rho^{ab}(2), \ldots, \rho^{ab}(n)$. □

4. Morphic Words

An infinite word w over some alphabet Σ is a *morphic word* if there exists an alphabet Γ and morphisms $\sigma : \Gamma^* \to \Sigma^*$ and $\tau : \Gamma^* \to \Gamma^*$, with σ uniform with morphic length 1, such that for some $a_0 \in \Gamma$, $w = \sigma(\tau^\omega(a_0))$. We call σ and τ the *underlying morphisms* of the morphic word. If τ is uniform with morphic length ℓ, then w is an *automatic sequence* and we say it has morphic length ℓ (or we say it is an ℓ-automatic sequence), e.g., if $\tau = (aaabc, aabca, bcbcc)$ and $\sigma = (0, 0, 1)$, then

$$\sigma(\tau^\omega(a)) = 000010000100000100010010011\cdots$$

is an 5-automatic sequence. This concept of morphic length is not uniquely defined for a given word: in particular, if the morphic length can be ℓ then it can be ℓ^n for every positive integer n by using τ^n in place of τ. Also, for simplicity we generally write 0 for the letter a_0 at which we are evaluating the fixed point.

Consider the two automatic sequences, with the same morphic length, constructed respectively from $\tau_1 = (001, 020, 221)$ and $\sigma_1 = (0, 1, 1)$ and from $\tau_2 = (012, 112, 200)$ and $\sigma_2 = (0, 1, 1)$. Combine them by adding the corresponding digits:

$$\begin{array}{r} 0010010100010010100011110 01\cdots \\ +\ 0111111001111111001000110 11\cdots \\ \hline 0121121101121121101011220 12\cdots . \end{array}$$

We claim that the resulting word over $\{0, 1, 2\}$ is an automatic sequence with the same morphic length. The following result, which establishes this claim, is a basic property of automatic sequences that we find useful later.

Proposition 3. *[1, Theorem 5.4.4] Let Σ, Γ, and Π be three alphabets. Let $u \in \Sigma^*$ and $v \in \Gamma^*$ be ℓ-automatic sequences. Let $f : \Sigma \times \Gamma \to \Pi$ be a function and w be the infinite word such that $w[i] = f(u[i], v[i])$, for all non-negative integers i. Then w is also an ℓ-automatic sequence.*

Proposition 3 does not necessarily hold if morphic words are replaced by fixed points of morphisms unless f is bijective. This is because after

combining the two morphic words, we might get one letter in two different ways, and so some factors of the fixed point that should be the same because they are generated by the same letter may in fact be different, and so the resulting word may not be a fixed point of a morphism. Proposition 4 is also interesting because it shows that, unlike fixed points of morphisms, automatic sequences are closed under the addition or removal of prefixes. It is stated for automatic sequences, but it can be stated to apply to general morphic words as long as the underlying morphism τ is non-erasing.

Proposition 4. *Let u be an ℓ-automatic sequence.*

1. *If v is a word obtained from u by adding any finite word (even one with letters not occurring in u) to the beginning of u, then v is an ℓ-automatic sequence.*

2. *If v is a word obtained from u by removing a prefix of u, then v is an ℓ-automatic sequence.*

5. Structure in the Abelian Complexity for Binary Uniform Morphisms

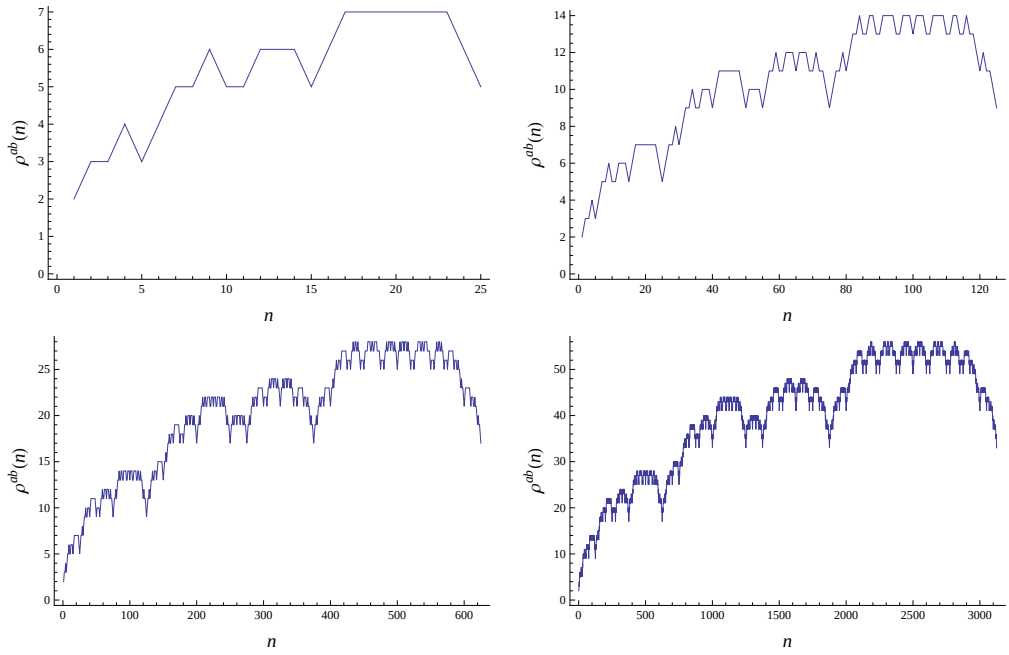
Figure 1 suggests that knowing the first n values, $1, 2, \ldots, n$, of the abelian complexity for a fixed point of a binary uniform morphism with morphic length ℓ gives us the values for $\ell, 2\ell, \ldots, n\ell$, which is why we see a type of self-similarity; this result was shown in [4]. However, there is much more structure; between the multiples of ℓ, the same types of repeating pattern occur. Trying to describe these yields the following result regarding the structure of z_m and z_M.

Before proving it, we recall the notations from Section 2 of the difference of a binary uniform morphism φ, denoted d, which is $|z_0 - z_1|$, where $z_0 = z(\varphi(0))$, i.e., the number of zeroes in the image of 0, and $z_1 = z(\varphi(1))$. We also use the standard notation Δf to denote the finite forward difference of f, i.e., $\Delta f(n) = f(n+1) - f(n)$.

Theorem 5. *If φ is a binary uniform morphism with morphic length ℓ and $d \neq 1$, then Δz_m and Δz_M are ℓ-automatic sequences.*

Proof. As in Section 3, assume that $z(\varphi(0)) = z_0 \geq z_1 = z(\varphi(1))$ (in case $z_0 < z_1$ the square of the morphism is considered, so that the length ℓ of the morphism changes to ℓ^2).

Figure 1: Self-similarity, from left-to-right and then top-to-bottom: the first 5^2, 5^3, 5^4, and 5^5 values of the abelian complexity for morphism $(00101, 11001)$ with morphic length 5.



Suppose $d = z_0 - z_1 > 1$ (the case $d = 0$ follows a similar but slightly simpler argument, and will be treated later). Call the table of the four lists with infinitely many rows produced by running Algorithm 1 forever on input φ, after any blank entry has been replaced by an X character, the z_m table. In the computations described below, ignore the X's. Now, create an infinite word w as follows. Set $w[0] = a_0$ and $w[1] = a_1$, where a_0 and a_1 are two special characters. If $j \geq 2$, then to compute $w[j]$, take rows j, $j+1$, and $j+2$ from the z_m table and place them into a 3×4 table with row j first and row $j+2$ last. Then, subtract the minimum value in this 3×4 table from all its entries, so that at least one entry is 0. Finally, if any entry in this 3×4 table exceeds 2ℓ, replace it with X. This gives us a 3×4 table whose entries are in $[0..2\ell] \cup \{X\}$, and set the letter $w[j]$ to be this table. Let the alphabet Γ be the set of all letters occurring in w.

We claim that w is the fixed point of the ℓ-uniform morphism τ defined over Γ as follows. Let $\tau(a_0) = w[0]w[1] \cdots w[\ell - 1]$ and $\tau(a_1) = w[\ell]w[\ell + 1] \cdots w[2\ell - 1]$. Now let a be any letter in Γ other than a_0 and a_1, and let n be such that $a = w[n]$. The letter a is a 3×4 table produced by taking rows n, $n+1$, and $n+2$ from the z_m table, subtracting out the minimum value, and replacing any value larger than 2ℓ with X. Use Algorithm 1 to compute what would be rows 2ℓ through $3\ell + 1$ of the z_m table, a total of $\ell + 2$ rows, under the assumption that rows 2, 3, and 4 are given by the rows of the table associated with a. Let $\tau(a)[i]$, for $i \in [0..\ell - 1]$, be obtained using rows $2\ell + i$, $2\ell + i + 1$, and $2\ell + i + 2$, subtracting the minimum value, and replacing all values larger than 2ℓ with X.

Note that when using Algorithm 1 for computing row $j\ell + r$ with $0 < r \leq \ell$, we recurse on row $j + 1$ if $r = 1$ and rows $j + 1$ and $j + 2$ if $r > 1$. To see this when $r > 1$ for instance, the only values congruent to $j\ell + r \pmod{\ell}$ in the range $[2..2\ell]$ are r and $\ell + r$, so we recurse on row $m = \frac{j\ell+r-r}{\ell} + 2 = j + 2$ or on row $m = \frac{(j\ell+r)-(\ell+r)}{\ell} + 2 = j + 1$. Therefore, when computing rows $j\ell$ through $(j + 1)\ell + 1$, we recurse on rows j, $j + 1$, and $j + 2$.

We next show that $\tau(w[n]) = w[n\ell] \cdots w[n\ell + \ell - 1]$. First, assume that we skip the step where we replace all entries larger than 2ℓ by X. We claim that the $\ell + 2$ rows produced by the above process are equivalent to rows $n\ell$ through $(n + 1)\ell + 1$ if we add some constant to every entry. The two aberrations between what we do to produce the $\ell + 2$ rows 2ℓ trough $3\ell + 1$ and what Algorithm 1 actually does to produce rows $n\ell$ through $(n+1)\ell+1$ are first, that the three rows used in the computation of $w[n]$ only differ from

the actual rows n, $n+1$, and $n+2$ of the z_m table by some constant C, and second, that we define the indices of the rows we recurse on to be 2, 3, and 4 instead of n, $n+1$, and $n+2$. The first aberration changes all entries in the rows produced by Cd, since Algorithm 1 is told that there are C fewer zeroes in each word it recurses on, which tells it that there are Cd fewer zeroes in the words produced, since we apply an iteration of the morphism φ (as $z_0 \geq z_1$, $d = z_0 - z_1$). The second aberration also affects all the entries by a constant, since Algorithm 1 thinks that the words being recursed on have $n-2$ fewer letters than they actually do, which makes it think that there are $n-2$ fewer ones in these words (since the number of zeroes is given), and therefore the entries produced are smaller by $(n-2)z_1$. Now, the letter $w[n\ell+i]$ is computed the same way as the letter $\tau(a)[i]$ except using rows $n\ell+i$, $n\ell+i+1$, and $n\ell+i+2$ instead of rows $2\ell+i$, $2\ell+i+1$, and $2\ell+i+2$. However, since we always subtract the minimum value, it does not matter if the entries in the rows used differ by a constant, so $\tau(a)[i] = w[n\ell+i]$ as desired.

Second, assume that we do not skip the step where we replace all entries larger than 2ℓ by X. If allowing this step changes any of the letters in $\tau(a)$ from what they were before, then it has changed some of the $\ell+2$ rows that we use Algorithm 1 to produce. If an entry in one of these rows has changed, then the only way for that entry to achieve its optimal value is for it to recurse on one of the values that we replace with X. Consider such an entry in row j, $j \in [2\ell..3\ell+1]$. Let one of the values that it recurses on to get its optimal value be Y in row k, $k \in [2..4]$, and let the smallest entry in row k be Z. Because Y gets replaced by X if we allow that step, $Y > Z + 2\ell$. Note that in row j, at least one entry is allowed to recurse on entry Z, because if v is a word corresponding to entry Z, there exists a factor of $\varphi(v)$ of length j that includes letters from every block of length ℓ, and such a word can be produced by recursing on entry Z. Any entry in row j that is allowed to recurse on entry Z is at most $(k-2)z_1 + Zd + 2\ell$, since in the worst case the border blocks contain 2ℓ zeroes. The entry whose optimal value is obtained from recursing on Y is at least $(k-2)z_1 + Yd$. Therefore, the difference between these entries is at least $(Y-Z)d - 2\ell > (2\ell)\cdot 2 - 2\ell = 2\ell$ since $Y - Z > 2\ell$ and $d > 1$. Thus, any entry that is changed by allowing that step exceeds another value in its row by more than 2ℓ, and so is replaced by X in the end anyway. Thus, allowing this step has no effect on $\tau(a)$.

Note that the part of the proof where we require $d > 1$ is in showing that we can delete all values larger than 2ℓ in a letter, which is a 3×4 table, to

make the alphabet size finite.

We have constructed τ in such a way that

$$\tau(w[n]) = w[n\ell]w[n\ell+1]\cdots w[n\ell+\ell-1]$$

for all $n \geq 2$, and by definition this holds for $n = 0$ and $n = 1$. Thus, $w = \tau^\omega(a_0)$.

Now, we wish to define a uniform morphism σ with morphic length 1 such that $\sigma(w[n]) = \Delta z_m[n]$ for all $n \geq 0$. Set $\sigma(a_0) = \Delta z_m[0]$ and $\sigma(a_1) = \Delta z_m[1]$, so that these values clearly satisfy the desired property. Then, for all $a \in \Gamma$ with $a \notin \{a_0, a_1\}$, let $\sigma(a)$ be the minimum value in the second row of a minus the minimum value in the first row of a. Because the three rows in $a = w[n]$ differ by some constant from rows n, $n+1$, and $n+2$ of the z_m table, $\sigma(w[n]) = z_m(n+1) - z_m(n) = \Delta z_m[n]$, as desired. Thus, $\sigma(\tau^\omega(a_0)) = \sigma(w) = \Delta z_m$, showing that Δz_m is an ℓ-automatic sequence.

Suppose $d = z_0 - z_1 = 0$. This case follows an argument similar to the $d > 1$ case but is slightly simpler. To construct the letters, instead of replacing any value larger than 2ℓ with X, we replace any value larger than 1 with a 1. The morphism τ still works, because when recursing in the $d = 0$ case we do not care about the actual values in the row we recurse on (since the number of zeroes in $\varphi(v)$ is $z_0|v|$, which is independent of v), but only whether each entry contains an integer or an X. This still is enough information to determine $\Delta z_m[n]$, since replacing any value larger than 1 with a 1 does not affect the difference between the minimum values in the first two rows of a letter $w[n]$. □

If we want to determine all the letters that occur in the fixed point $\tau^\omega(a_0)$ from the proof of Theorem 5, we can start with the set $S_0 = \{a_0, a_1\}$, then define $S_{n+1} = S_n \cup \{\text{the letters in } \tau(a) \mid a \in S_n\}$, and keep computing S_n for the next value of n until we get $S_{n+1} = S_n$ for some n. If a non-trivial fraction of the letters in the alphabet Γ defined in the proof actually occurs in the fixed point, this takes impractically long, but we have found that in practice the alphabet size is usually not very large, e.g., it is often around 20 for morphisms with $\ell \approx 5$.

Also when $d = 1$, we have not been able to make the alphabet size, in the proof of Theorem 5, finite. WHowever, for many morphisms satisfying $d = 1$, there are only finitely many letters in the fixed point of τ starting at a_0 even if we skip the step where we delete entries larger than 2ℓ, and so the proof works for them. To see if the proof works for a given morphism

with $d = 1$, we can apply the algorithm in the previous paragraph to find all letters that occur. If it terminates in a specified number of iterations, the proof works for that morphism, but if it does not terminate in the allowed number of iterations, then the algorithm is inconclusive.

Corollary 1. *If φ is a binary uniform morphism with morphic length ℓ and $d \neq 1$, the infinite word $\Delta\rho^{ab}$ over $\{-1, 0, 1\}$ is an ℓ-automatic sequence.*

Proof. By Theorem 5, Δz_m and Δz_M are ℓ-automatic sequences. Since $\rho^{ab}(n) = z_M(n) - z_m(n) + 1$ by Lemma 1, it follows that $\Delta\rho^{ab}[n] = \Delta z_M[n] - \Delta z_m[n]$. Now apply Proposition 3 to $\Delta\rho^{ab}$ to get the desired result. □

For example, consider the morphism $(00101, 11011)$. We can calculate the underlying morphisms for Δz_m and Δz_M from the proof of Theorem 5, and combine them to produce underlying morphisms for $\Delta\rho^{ab}$. Applying a simplification algorithm (Algorithm 8, which we discuss later) gives the following morphisms σ and τ to construct $\Delta\rho^{ab}$:

	$\sigma()$	$\tau()$		$\sigma()$	$\tau()$		$\sigma()$	$\tau()$		$\sigma()$	$\tau()$
a	1	$aabcd$	b	0	$eabfg$	c	1	$aahib$	d	-1	$fihgg$
e	0	$abefg$	f	0	$abfge$	g	-1	$cdfgg$	h	0	$eahge$
i	0	$cihid$									

6. Self-similarity

We prove some results on self-similarity for Δz_m and $\Delta\rho^{ab}$.

Proposition 6. *For a binary uniform morphism φ with morphic length ℓ (with $z_0 \geq z_1$), let $\sigma : \Gamma^* \to \{0, 1\}^*$ and $\tau : \Gamma^* \to \Gamma^*$ be uniform morphisms with morphic length 1 and ℓ, respectively, such that $\sigma(\tau^\omega(a_0)) = \Delta z_m$ for some letter a_0. For any integer $n > 0$, consider $a = \tau^\omega(a_0)[n]$. If $\sigma(a) = 0$ (resp., $\sigma(a) = 1$), then exactly z_1 (resp., z_0) occurrences of letters in $\tau(a)$ are mapped to 1 under σ. The same result holds for z_M.*

Proof. For any $n > 0$, consider the letter $a = \tau^\omega(a_0)[n]$, so that $\sigma(a) = \Delta z_m[n]$. Then in the fixed point of τ at a_0, a maps to the letters in positions included in $[\ell n..\ell(n + 1))$, so

$$\sigma(\tau(a)) = \Delta z_m[\ell n]\Delta z_m[\ell n + 1] \cdots \Delta z_m[\ell n + \ell - 1].$$

Also,

$$\Delta z_m[\ell n] + \Delta z_m[\ell n + 1] + \cdots + \Delta z_m[\ell n + \ell - 1] = z_m(\ell(n+1)) - z_m(\ell n).$$

Since σ maps every letter to 0 or 1, the number of occurrences of letters in $\tau(a)$ that are mapped to 1 under σ is $z_m(\ell(n+1)) - z_m(\ell n)$. Since $n > 0$, we can apply [4, Lemma 13] to get

$$z_m(\ell(n+1)) - z_m(\ell n) = d(z_m(n+1) - z_m(n)) + z_1 = (z_0 - z_1)\Delta z_m(n) + z_1.$$

When $\Delta z_m(n) = 0$, this is z_1, and when $\Delta z_m(n) = 1$, this is z_0. Recalling that $\sigma(a) = \Delta z_m[n]$, this shows that when $\sigma(a) = 0$, exactly z_1 occurrences of letters in $\tau(a)$ are mapped to 1 under σ, and when $\sigma(a) = 1$, exactly z_0 occurrences of letters in $\tau(a)$ are mapped to 1 under σ. The proof is similar for z_M. □

Observe that when $n = 0$, $z_m(\ell) - z_m(0) = dz_m(1) + z_1 - \Delta$ by [4, Lemma 13], where $\Delta = \min\{z_0, z_1\} - z_m(\ell)$ by [4, Lemma 10]. When $\varphi(0)$ contains only zeroes, clearly $\sigma(b) = 1$ for every letter b in Γ (since $z_m(i) = i$ for all $i \geq 0$), so the ℓ occurrences of letters in $\tau(a_0)$ are mapped to 1 under σ. Otherwise, $z_m(1) = 0$, so $\sigma(a_0) = z_m(1) - z_m(0) = 0$, and exactly $z_1 - \Delta$ occurrences of letters in $\tau(a_0)$ are mapped to 1 under σ. For z_M, a similar derivation from [4, Lemma 13] shows that exactly $z_0 + \Delta$ occurrences of letters in $\tau(a_0)$ are mapped to 1 under σ.

Proposition 6 is significant for a few reasons. First, the above observations show that when $\Delta \neq 0$, the letter a_0 occurs only once in the respective fixed points of the underlying morphisms τ for Δz_m and Δz_M starting at a_0. Second, it confirms that the difference $z_m(1) - z_m(0)$ is a good consecutive difference to start at, because if for example we define our infinite word Δz_m to start with the difference $z_m(2) - z_m(1)$ instead, then this result does not hold. Third, it shows that Δz_m has a sort of self-similarity: the number of occurrences of the letters in $\tau(a)$, $a \neq a_0$, corresponds to a bigger increase, i.e., z_0, if a itself does correspond to an increase (when $\sigma(a) = 1$), and a smaller increase, i.e., z_1, if a does not (when $\sigma(a) = 0$). In fact, this self-similarity holds in a more obvious way for $\Delta\rho^{ab}$, as the following result demonstrates.

Proposition 7. *If φ is a binary uniform morphism with morphic length ℓ (with $z_0 \geq z_1$) and $n > 0$ is an integer, then $\rho^{ab}(\ell(n+1)) - \rho^{ab}(\ell n) = d(\Delta\rho^{ab}(n))$.*

Proof. By [4, Lemma 15], $\rho^{ab}(\ell n) = d\rho^{ab}(n) - d + 1 + 2\Delta$, where $\Delta = \min\{z_0, z_1\} - z_m(\ell)$. The desired equality follows. □

7. Eliminating Redundancies in Automatic Sequences

Even if two letters have the same image under a morphism $\tau : \Gamma^* \to \Gamma^*$, they appear as "distinct" in a fixed point of a morphism with respect to τ. However, in an automatic sequence with respect to two morphisms $\sigma : \Gamma^* \to \Sigma^*$ (with morphic length 1) and $\tau : \Gamma^* \to \Gamma^*$ (with morphic length ℓ), it is possible that two letters have the same image under τ *and* are mapped to the same letter under σ. Thus these two letters appear as "identical" for our purposes, and it would be nice to eliminate one of them to simplify σ and τ.

We say that two letters a and b in Γ are *equivalent* or *redundant* with respect to the morphisms σ and τ if for all integers $n \geq 0$, we have $\sigma(\tau^n(a)) = \sigma(\tau^n(b))$. Note that if two letters are equivalent, then replacing any instance of one by the other in any image of a letter under τ does not change the automatic sequence produced. Determining whether two letters are equivalent is not as simple as checking whether their images under σ and τ are the same. This is clearly sufficient to imply equivalence. But it is possible that two equivalent letters a and b have nearly the same image under τ, except that in a position i, $\tau(a)[i]$ and $\tau(b)[i]$ are distinct but equivalent letters themselves. Furthermore, it is possible to have equivalent letters even if no two letters have the same image under τ, so the problem cannot be solved by repeatedly merging letters that have the same image under τ.

The following remarks motivate an algorithm for eliminating redundant letters. For any pair of non-equivalent letters a, b, let $f(a, b)$ be the smallest integer n such that $\sigma(\tau^n(a)) \neq \sigma(\tau^n(b))$. It is trivial to determine all pairs (a, b) with $f(a, b) = 0$; this occurs if and only if $\sigma(a) \neq \sigma(b)$. Now fix an integer n, and assume we have determined all pairs of non-equivalent letters (a, b) with $f(a, b) < n$. Pick any two distinct letters a and b. Let $\tau(a) = a_0 a_1 \cdots a_{\ell-1}$ and $\tau(b) = b_0 b_1 \cdots b_{\ell-1}$. Then

$$\sigma(\tau^n(a)) = \sigma(\tau^{n-1}(a_0))\sigma(\tau^{n-1}(a_1)) \cdots \sigma(\tau^{n-1}(a_{\ell-1}))$$

and

$$\sigma(\tau^n(b)) = \sigma(\tau^{n-1}(b_0))\sigma(\tau^{n-1}(b_1)) \cdots \sigma(\tau^{n-1}(b_{\ell-1})).$$

We have $f(a, b) \leq n$ if and only if $\sigma(\tau^{n-1}(a_i)) \neq \sigma(\tau^{n-1}(b_i))$ for some i if and only if $f(a_i, b_i) < n$. Therefore, we can determine all pairs (a, b)

with $f(a,b) = n$ if we know all pairs (a,b) with $f(a,b) < n$. Furthermore, if there are no pairs (a,b) with $f(a,b) = n$, then there are no pairs (a,b) with $f(a,b) = n+1$; otherwise, for some pair (a,b) we have $\sigma(\tau^{n+1}(a)) \neq \sigma(\tau^{n+1}(b))$ implying that for some i, $f(\tau(a)[i], \tau(b)[i]) = n$, a contradiction.

Algorithm 8. *Given as input uniform morphisms $\sigma : \Gamma^* \to \Sigma^*$ (with morphic length 1) and $\tau : \Gamma^* \to \Gamma^*$ (with morphic length ℓ), produce two uniform morphisms $\sigma' : \Pi^* \to \Sigma^*$ (with morphic length 1) and $\tau' : \Pi^* \to \Pi^*$ (with morphic length ℓ) as follows:*

- *Iteration 0: partition the letters in Γ into equivalence classes based on the value of $\sigma(\tau^0(a))$ for every letter a in Γ, i.e., the class of a, $[a]$, is $\{b \in \Gamma \mid \sigma(a) = \sigma(\tau^0(a)) = \sigma(\tau^0(b)) = \sigma(b)\}$. Assign each $[a]$ a unique ID, denoted by $ID([a])$.*

- *Iteration n, $n > 0$: within each $[a]$, for each letter b in $[a]$, compute the sequence of IDs, $ID([b_0])ID([b_1]) \cdots ID([b_{\ell-1}])$, generated by replacing the letters in $\tau(b) = b_0 b_1 \cdots b_{\ell-1}$ by the IDs of those letters' classes. Then split each $[a]$ into new equivalence classes based on those sequences of IDs. After doing this for every equivalence class, get rid of the old IDs and assign an ID to each equivalence class.*

- *Repeat this process until no new equivalence classes are created (corresponding to no letters $a, b \in \Gamma$ satisfying $f(a,b) = n$ if this is iteration n).*

- *Assign each equivalence class a letter, say A, which is the set of all letters in the class (Π is the set of all those A's). Find $\sigma'(A)$ by taking the image under σ of any of the letters in A and find $\tau'(A)$ by taking any $a \in A$, computing $\tau(a)$, and replacing each letter in $\tau(a)$ with the letter assigned to its equivalence class.*

Theorem 9. *Given as input uniform morphisms $\sigma : \Gamma^* \to \Sigma^*$ (with morphic length 1) and $\tau : \Gamma^* \to \Gamma^*$ (with morphic length ℓ), Algorithm 8 outputs two uniform morphisms $\sigma' : \Pi^* \to \Sigma^*$ (with morphic length 1) and $\tau' : \Pi^* \to \Pi^*$ (with morphic length ℓ), where alphabet Π is possibly smaller than alphabet Γ and no equivalent letters exist in Π. It runs in $\mathcal{O}(|\Gamma|^2 \ell)$ time.*

Proof. We first claim that for all non-equivalent letters $a, b \in \Gamma$, the inequality $f(a,b) < |\Gamma| - 1$ holds. To show our claim, it is sufficient to note that

the number of classes is bounded by $|\Gamma|$ and each step increases by at most 1 this number.

Our claim shows that Algorithm 8 goes through at most $|\Gamma|-1$ iterations. If we use a hash table to store equivalence class IDs, then for each iteration we compute the ID sequence for the image of each letter under τ, which takes $|\Gamma|\ell$ time. Computing σ' and τ' thus takes $\mathcal{O}(|\Gamma|\ell)$ time. □

For example, given as input the morphisms $\sigma : \{a,b,c,d,e,f\}^* \to \{0,1,2\}^*$ and $\tau : \{a,b,c,d,e,f\}^* \to \{a,b,c,d,e,f\}^*$ given by $\sigma = (0,1,1,2,2,2)$ and $\tau = (ab, bd, ce, ab, ac, ad)$, Algorithm 8 outputs the morphisms $\sigma' : \{w,x,y,z\}^* \to \{0,1,2\}^*$ and $\tau' : \{w,x,y,z\}^* \to \{w,x,y,z\}^*$ with no equivalent letters given by $\sigma' = (0,1,2,2)$ and $\tau' = (wx, xy, wx, wy)$. At iteration 0, the alphabet $\{a,b,c,d,e,f\}$ is partitioned into the equivalence classes $\{a\}, \{b,c\}, \{d,e,f\}$. At iteration 1, the class $\{d,e,f\}$ is split into two classes $\{d,e\}, \{f\}$. At iteration 3, no more classes are created. So assign the letter w to the class $\{a\}$, x to $\{b,c\}$, y to $\{d,e\}$, and z to $\{f\}$.

8. Remarks and Open Problems for Future Work

The concept of abelian complexity was recently extended to the one of k-abelian complexity [21]: two finite words u and v are *k-abelian equivalent* if for all words x of length at most k, the number of occurrences of x in u equals that number in v.

Parreau et al. [25] developed a new approach for studying the k-abelian complexity of 2-automatic sequences, which is based on the fact that a sequence satisfying some reflection symmetry property ends up being 2-regular, i.e., the $\mathbb{Z}$-module generated by its 2-kernel is finitely generated. They proved, in particular, that the period-doubling word and the Thue-Morse word have 2-abelian complexity sequences that are 2-regular and that the 2-block codings of these two words have 1-abelian complexity sequences that are 2-regular. In fact, they conjectured that any ℓ-automatic sequence has an k-abelian complexity function that is ℓ-regular (the case of $k = 1$ appears as an open problem in [29], and it was raised as a question in [22] where it was shown true for the paper-folding word). Charlier et al. [10] showed that any ℓ-automatic sequence has a factor complexity function that is ℓ-regular, and gave an algorithmic method to compute it.

In this paper, we deal with 1-abelian complexity (or abelian complexity) of sequences constructed from binary ℓ-uniform morphisms. We are not

directly interested in the ℓ-regularity of the abelian complexity sequences of this class of ℓ-automatic sequences; we develop an $\mathcal{O}(n)$-time algorithm to compute the abelian complexity for all indices from 1 up to n. Using our algorithm we also analyze the structure in the abelian complexity for that class of sequences.

Analyzing the abelian complexity values of various binary uniform morphisms in depth suggested the following conjecture, for which the $d = 1$ case remains open (see Theorem 5 and Corollary 1 for the $d \neq 1$ case).

Conjecture 10. *If φ is a binary uniform morphism with morphic length ℓ, then Δz_m, Δz_M, and $\Delta\rho^{ab}$ are ℓ-automatic sequences.*

Also, regarding the underlying morphisms τ for the automatic sequences Δz_m, Δz_M, and $\Delta\rho^{ab}$, we have seen in practice that the simplified alphabet sizes are relatively small, typically less than ℓ. An open problem is to give a bound for these sizes.

One of the largest ratios of the alphabet size to ℓ we have encountered was for the morphism $\varphi = (0011100, 1100011)$ with $\ell = 7$. In this case Δz_m, Δz_M, and $\Delta\rho^{ab}$ all have simplified alphabet size 17 for the morphism τ. Note that $d = 1$ for φ, but we were able to determine that Conjecture 10 held for it by using the parts of the proof of Theorem 5 that apply to all values of d, and then noting by casework that we can replace any value exceeding 5 with an X (like how we show that we can replace values larger than 2ℓ with an X). Using Algorithm 8, the simplified morphisms σ and τ for $\Delta z_m = \sigma(\tau^\omega(a))$ for φ are as follows:

	$\sigma()$	$\tau()$		$\sigma()$	$\tau()$		$\sigma()$	$\tau()$
a	0	$abbcdef$	g	1	$jehkdef$	m	1	$ghijmef$
b	0	$ghbijef$	h	0	$ghbcdef$	n	0	$gpdehbn$
c	0	$gfdehbi$	i	0	$gfdehbn$	o	1	$omehkqf$
d	1	$jefgfgf$	j	1	$omehkgf$	p	0	$ghcdehi$
e	1	$ghklmef$	k	0	$gfgfghi$	q	1	$jehklef$
f	0	$ghkdehi$	l	1	$jmehkgf$			

Another observation we have made is that for many morphisms, $\Delta z_m(n) + \Delta z_M(n) = 1$ for all $n \geq 0$. This means that from n to $n + 1$, exactly one of z_m and z_M increases, which means that $\Delta\rho^{ab}(n)$ is always -1 or 1. An open problem is to identify the class of morphisms for which this property holds. The above φ is an example of such a morphism.

Finally, our results apply almost exclusively to binary uniform morphisms, but we have seen that there seems to be definite structure in the abelian complexity of non-binary uniform morphisms. Another open problem for future work is to find ways to efficiently compute the abelian complexity values in these cases, and to analyze the mathematical structure and asymptotic complexity of the abelian complexity function.

Acknowledgements

Project sponsored by the National Security Agency under Grant Number H98230-15-1-0232. The United States Government is authorized to reproduce and distribute reprints notwithstanding any copyright notation herein. This manuscript is submitted for publication with the understanding that the United States Government is authorized to reproduce and distribute reprints. This material is based upon work supported by the National Science Foundation under Grant No. DMS–1060775.

We thank Benjamin De Winkle as well as the referees of preliminary versions of this paper for their very valuable comments and suggestions.

A research assignment from the University of North Carolina at Greensboro for the first author is also gratefully acknowledged. Some of this assignment was spent at the IRIF: Institut de Recherche en Informatique Fondamentale of Université Paris-Diderot–Paris 7, France.

References

[1] J.-P. Allouche and J. Shallit. Automatic Sequences: Theory, Applications, Generalizations. Cambridge University Press, 2003.

[2] L. Balková, K. Břinda, and O. Turek. Abelian complexity of infinite words associated with quadratic Parry numbers. *Theoret. Comput. Sci.*, 412:6252–6260, 2011.

[3] F. Blanchet-Sadri, J. D. Currie, N. Fox, and N. Rampersad. Abelian complexity of fixed point of morphism $0 \to 012, 1 \to 02, 2 \to 1$. *Integers*, 14:#A11, 2014.

[4] F. Blanchet-Sadri, N. Fox, and N. Rampersad. On the asymptotic abelian complexity of morphic words. *Adv. Appl. Math.*, 61:46–84, 2014.

[5] S. Böcker. Sequencing from compomers: Using mass spectrometry for DNA de novo sequencing of 200+nt. *J. Comput. Biol.*, 11:1110–1134, 2004.

[6] S. Böcker. Simulating multiplexed SNP discovery rates using base-specific cleavage and mass spectrometry. *Bioinformatics*, 23:5–12, 2007.

[7] P. Burcsi, F. Cicalese, G. Fici, and Z. Lipták. Algorithms for jumbled pattern matching in strings. *Internat. J. Found. Comput. Sci.*, 23:357–374, 2012.

[8] P. Burcsi, F. Cicalese, G. Fici, and Z. Lipták. On approximate jumbled pattern matching in strings. *Theory Comput. Syst.*, 50:35–51, 2012.

[9] A. Butman, R. Eres, and G. M. Landau. Scaled and permuted string matching. *Inform. Process. Lett.*, 92:293–297, 2004.

[10] E. Charlier, N. Rampersad, and J. Shallit. Enumeration and decidable properties of automatic sequences. *Internat. J. Found. Comp. Sci.*, 23:1035–1066, 2012.

[11] M. Christodoulakis and M. Christou. Abelian concepts in strings: a review. In J. Holub, B. W. Watson, and J. Zdarek, editors, *Festschrift for Borivoj Melichar*, pages 19–45. Czech Technical University in Prague, Czech Republic, 2012.

[12] E. M. Coven and G. A. Hedlund. Sequences with minimal block growth. *Math. Syst. Theory*, 7:138–153, 1973.

[13] J. Currie and N. Rampersad. Recurrent words with constant abelian complexity. *Adv. Appl. Math.*, 47:116–124, 2011.

[14] A. Ehrenfeucht, K. P. Lee, and G. Rozenberg. Subword complexities of various classes of deterministic developmental languages without interactions. *Theoret. Comput. Sci.*, 1:59–75, 1975.

[15] A. Ehrenfeucht and G. Rozenberg. On the subword complexity of D0L languages with a constant distribution. *Inform. Process. Lett.*, 13:108–113, 1981.

[16] A. Ehrenfeucht and G. Rozenberg. On the subword complexity of square-free D0L languages. *Theoret. Comput. Sci.*, 16:25–32, 1981.

[17] A. Ehrenfeucht and G. Rozenberg. On the subword complexity of locally catenative D0L languages. *Inform. Process. Lett.*, 16:121–124, 1983.

[18] R. Eres, G. M. Landau, and L. Parida. Permutation pattern discovery in biosequences. *J. Comput. Biol.*, 11:1050–1060, 2004.

[19] A. Evdokimov and S. Kitaev. Crucial words and the complexity of some extremal problems for sets of prohibited words. *J. Combin. Theory Ser. A*, 105:273–289, 2004.

[20] A. E. Frid. The subword complexity of fixed points of binary uniform morphisms. In *FCT 1997*, vol. 1279 of *LNCS*, pages 179–187, Berlin, Heidelberg, 1997. Springer-Verlag.

[21] J. Karhumäki, A. Saarela, and L. Q. Zamboni. On a generalization of abelian equivalence and complexity of infinite words. *J. Combin. Theory Ser. A*, 120:2189–2206, 2013.

[22] B. Madill and N. Rampersad. The abelian complexity of the paperfolding word. *Discrete Math.*, 313:831–838, 2013.

[23] J.-J. Pansiot. Complexité des facteurs des mots infinis engendrés par morphismes itérés. In *ICALP 1984*, vol. 172 of *LNCS*, pages 380–389, Berlin, Heidelberg, 1984. Springer-Verlag.

[24] L. Parida. Gapped permutation patterns for comparative genomics. In *WABI 2006*, pages 376–387, 2006.

[25] A. Parreau, M. Rigo, E. Rowland, and E. Vandomme. A new approach to the 2-regularity of the ℓ-abelian complexity of 2-automatic sequences. *Electr. J. Comb.*, 22:P1.27, 2015.

[26] G. Richomme, K. Saari, and L. Q. Zamboni. Balance and abelian complexity of the Tribonacci word. *Adv. Appl. Math.*, 45:212–231, 2010.

[27] G. Richomme, K. Saari, and L. Q. Zamboni. Abelian complexity in minimal subshifts. *J. Lond. Math. Soc.*, 83:79–95, 2011.

[28] A. Saarela. Ultimately constant abelian complexity of infinite words. *J. Autom. Lang. Comb.*, 14:255–258, 2010.

[29] J. Shallit. Enumeration and automatic sequences. *PU. M. A.*, 25:96–106, 2015.

[30] O. Turek. Abelian properties of Parry words. *Theoret. Comput. Sci.*, 566:26–38, 2015.