Observations On Compressed Pattern-Matching with Ranked Variables in Zimin Words

Radosław Głowinski[2] and Wojciech Rytter[1,2]

[1] Department of Mathematics, Computer Science and Mechanics,
University of Warsaw, Warsaw, Poland
rytter@mimuw.edu.pl

[2] Faculty of Mathematics and Informatics,
Nicolaus Copernicus University, Toruń, Poland
glowir@mat.umk.pl

Abstract. Zimin words are very special finite words which are closely related to the pattern-avoidability problem. This problem consists in testing if an instance of a given pattern with variables occurs in almost all words over any finite alphabet. The problem is not well understood, no polynomial time algorithm is known and its NP-hardness is also not known. The pattern-avoidability problem is equivalent to searching for a pattern (with variables) in a Zimin word. The main difficulty is potentially exponential size of Zimin words. We use special properties of Zimin words, especially that they are highly compressible, to design efficient algorithms for special version of the pattern-matching, called here *ranked matching*. It gives a new interpretation of Zimin algorithm in compressed setting. We discuss the structure of rankings of variables and compressed representations of values of variables.

1 Introduction

The research on pattern avoidability started in late 70's in the papers by Bean, Ehrenfeucht and McNaulty [1], and independently by Zimin [5]. In the avoidability problem two disjoint finite alphabets, $A = \{a, b, c, \dots\}$ and $V = \{x_1, x_2, x_3, \dots\}$ are given, the elements of A are letters (constants) and the elements of V are variables. We denote the empty word by ε. A pattern π is a sequences of variables. The language of a pattern with respect to an alphabet A consists of words $h(\pi)$, where h is any non-erasing morphism from V^* to A^+. We say that word w encounters pattern π (or pattern occurs in this word) when there exists a morphism h, such that $h(\pi)$ is a subword of w. In the other case w avoids π.

The pattern π is unavoidable on A if every long enough word over A encounters π, otherwise it is avoidable on A. If π is unavoidable on every finite A then π is said to be unavoidable.

Example 1. The pattern $\alpha\alpha$ is avoidable over $A = \{a, b, c\}$. Let

$$u = abcacbabcbac\dots$$

be the infinite word generated by morphism

$$\mu : a \to abc, b \to ac, c \to b$$

starting from the letter a. The word u avoids the pattern $\alpha\alpha$ (u is square-free), as shown in [4]. Hence $\alpha\alpha$ is not unavoidable, however $\alpha\beta\alpha$ is unavoidable.

Radosław Głowinski, Wojciech Rytter: Observations On Compressed Pattern-Matching with Ranked Variables in Zimin Words, pp. 162–172.
Proceedings of PSC 2011, Jan Holub and Jan Žďárek (Eds.), ISBN 978-80-01-04870-2 © Czech Technical University in Prague, Czech Republic

The crucial role in avoidability problems play the words introduced by Zimin [5], called Zimin words, and denoted here by Z_k.

Definition 2. (of Zimin words) *Let*

$$Z_1 = 1,\ Z_k = Z_{k-1}\, k\, Z_{k-1}$$

Example 3. $Z_1 = 1,\ \ Z_2 = 121,\ \ Z_3 = 1213121,\ \ Z_4 = 121312141213121$

Observe that these words are exponentially long, however they have a very simple structure implying many useful properties. Define the Zimin morphism

$$\mu:\ 1 \to 121,$$

$$i \to i+1\ (\forall\, i > 1).$$

Fact 1

- *The morphism μ generates next Zimin word by mapping each letter according to μ. In other words: $Z_k = \mu(Z_{k-1})$.*
- *Each Zimin word, considered as a pattern, is unavoidable. Moreover it is a longest unavoidable pattern over k-th letter alphabet. There exists only one (up to letter permutation) unavoidable pattern of length $2^k - 1$ over a k-th letter alphabet and it is Z_k.*

The main property of Zimin words is that the avoidability problem is reducible to pattern-matching in Zimin words, see [5], [2].

Lemma 4. *π is an unavoidable pattern if and only if π occurs in Z_k, where k is the number of distinct symbols occurring in π.*

2 Compact representation of pattern instances

We say that a sequence u is j-interleaved iff for each two adjacent elements of u exactly one is equal to j.

The Zimin word Z_k can be alternatively defined as follows:

(A) Z_k starts with 1 and ends with 1;
(B) $|Z_k| = 2^k - 1$;
(C) For each $1 \leq j \leq k$ after removing all elements smaller than j the obtained sequence is j-interleaved.

Observation 1 *If we have a string $u \in \{1, 2, \ldots, k\}^+$, then u is a factor (subword) of Z_k iff it satisfies the condition (C).*

We omit the proof.

This gives a simple linear time algorithm to check if an explicitly given sequence is a factor of Z_k. However we are dealing with patterns, and instances of the pattern can be exponential with respect to the length of the pattern and the number of distinct variables. The instance is given by values of each variable which are factors of Z_k. Hence we introduce compact representation of factors of Zimin words.

We partition u into $w_1\, m\, w_2$, where m is a highest number in w (in every subword of Zimin word the highest number occurs exactly once). Then for each element i of w_1, respectively w_2, we remove i if there are larger elements to the left and to the right of this element. In other words if there is a factor $s\,\alpha\, i\, \beta\, t$, with $i < s$, $i < t$, we remove the element i. Denote by $compress(u)$ the result of removing all redundant i in u.

Observation 2 *compress(u) uniquely encodes a factor of a Zimin word.*

Example 5.

$$compress(21412131215 12131) \;=\; 24531$$

$$compress(Z_4) \;=\; compress(121312141213121) \;=\; 1234321$$

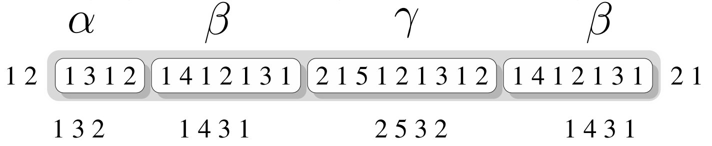

Figure 1. Example of a compact representation. In the first line there is a pattern, in the second uncompressed valuation of variables and in the third compressed valuation.

Fact 2 *For any $u \in \{1, 2, \ldots, k\}^+$ the first and the last elements of u and $compress(u)$ are respectively equal.*

Notice that compressed representation of any subword of Z_k has at most $2k-1$ letters and the representation of all variables requires $O(k^2)$ memory (under the assumption that only $O(1)$ space is necessary for representing each number).

For a valuation (morphism) h of the variables by its ranking function R_h we mean the function which assigns to each variable x_i its rank: $R_h(x_i)$ denotes the maximal letter in $h(x_i)$.

For a pattern π and a given valuation of variable by $\pi_{(i)}$ we mean the pattern with variables of ranks smaller than i removed from π.
By $\mathcal{V}_i(\pi)$ we define the set of variables from π with rank i.
For two strings u, w we write $u \le w$ iff u is a subword of w. By $\pi \rightarrow_h Z_k$ we mean that $h(\pi) \le Z_k$ and by $\pi \rightarrow Z_k$ we mean that for some morphism $h, h(\pi) \le Z_k$.

We extend the definition of j-interleaved sequence to patterns.

Definition 6. *Pattern π with compact representation of variables is j-interleaved iff for any two adjacent variables xy in π either x ends with j or y starts with j, in its compressed form.*

Theorem 7. *Assume we are given a pattern π of size n with k variables and a compact representation of values of the variables. Then we can check if the given compressed instance of π occurs in Z_k in time $O(nk)$.*

Proof. Assume we have an instance of the pattern $\pi = x_1 \cdots x_n$ with variables given in compressed form, such that for $1 \le j \le k$ π is j-interleaved when we remove all elements smaller than j from the compressed forms of the variables.

We define function red_i on strings over $\{1, 2, \dots, k\}$ which removes from the string all elements smaller than i. Performing red_i on patterns removes all elements smaller than i from the compact representations of the variables.

Using Observation 2 we obtain unique full representation of variables denoted $y_1 = val(x_1), \dots, y_n = val(x_n)$. Since y_i is a factor of a Zimin word $red_j(y_i)$ is j-interleaved. Because $red_j(\pi)$ is j-interleaved using Fact 2 we see that the concatenation $red_j(y_1)red_j(y_2)\cdots red_j(y_n)$ is j-interleaved (as a string). From Observation 1 we know that it is a factor of Z_k.

We showed that to determine if an instance of π occurs in Z_k we only need to check if for every $1 \le j \le k$ pattern $red_j(\pi)$ is j-interleaved. This can be done in total time $O(n\,k)$. □

3 Some properties of Zimin words and free sets

The ranking sequence associated with π is the sequence of ranks of consecutive variables in π.

Assume for a while that our pattern π is a permutation of n variables and we ask for the set of possible ranking sequences.

The ranking sequence has many useful properties:

1. Between every two occurrences of the same number a in ranking sequence there should be a number larger than a.
2. The ranking function is not necessarily injective (one to one).
3. If x_1x_2 and x_1x_3 are subwords of a pattern π, $x_1, x_2, x_3 \in V$ and $rank(x_3) < rank(x_2) < rank(x_1)$ and there exists a morphism φ that morphs p into Z_k then $\varphi(x_3)$ is a proper prefix of $\varphi(x_2)$.

Let $\overline{r}(\pi, h)$ be the set of ranks of variables in π for the valuation h. For example, for a pattern $\pi = \alpha\beta\alpha\gamma\beta\alpha$ and a valuation $h(\alpha) = 1, h(\beta) = 2, h(\gamma) = 31$ ranking sequence is $(1, 2, 1, 3, 2, 1)$ and $\overline{r}(\pi, h) = \{1, 2, 3\}$.

The following three facts are consequences of the proof of Zimin theorem (see [5] for details).

Lemma 8.

1. *If pattern π is unavoidable then there exists a morphism h such that $h(\pi)$ occurs in Z_k and* $\min \overline{r}(\pi, h) = 1$.
2. *If $\pi \to_h Z_k$ and* $\min \overline{r}(\pi, h) = j + 1 > 1$ *then there exists morphism g such that $\pi \to_g Z_k$ and $\overline{r}(\pi, g) = \{r - j : r \in \overline{r}(\pi, h)\}$.*
3. *If $\pi \to Z_k$ then there exists morphism h such that the set of ranks is an interval: $\overline{r}(\pi, h) = \{1, \dots, m\}$, for some $1 \le m \le k$.*

We present Zimin algorithm based on free sets and σ-deletions.

Definition 9. *$F \subseteq V$ is a* ***free set*** *for $\pi \in V^+$ if and only if there exist sets $A, B \subseteq V$ such that $F \subseteq B \setminus A$ where, for all $xy \le \pi, x \in A$ if and only if $y \in B$.*

Definition 10. *The mapping σ_F is a* ***σ-deletion*** *of π if and only if $F \subseteq V$ is a free set for π and $\sigma_F : V \to V \cup \{\varepsilon\}$ is defined by*

$$\sigma_F(x) = \begin{cases} x & \textit{if } x \notin F \\ \varepsilon & \textit{if } x \in F \end{cases}$$

The proof of the following fact can be found in [3], Zimin's algorithm is based on this fact.

Lemma 11. *π is an unavoidable pattern if and only if π can be reduced to ε by a sequence of σ-deletions.*

Unfortunately it is insufficient to remove only singleton free sets. There are patterns, which require the removing more than one element free sets, for example the pattern

$$\alpha\beta\alpha\gamma\alpha'\beta\alpha\gamma\alpha\beta\alpha'\gamma\alpha'\beta\alpha'$$

Therefore we can have exponentially many choices for free sets.

Lemma 12. *If $\pi \to Z_k$ then $R_1(\pi)$ is a free set.*

Proof. To satisfy the definition of a free set we need to give sets A and B, such that $R_1(\pi) \subset B \setminus A$ and all predecessors of variables from B are in A, all successors of variables from A are in B. We put all variables starting with 1 as the set B and all variables that do not end with 1 as set A. □

Lemma 13. *If $\pi \to Z_k$ then $\pi_{(2)} \to Z_{k-1}$.*

Proof. If $\pi \to Z_k$ then there exists morphism h, such that $h(\pi)$ is a subword of Z_k. $\mathcal{V}_1(\pi)$ is a set of variables x, such that $h(x) = 1$. We shall notice that if we remove all 1 from Z_k we obtain Z_{k-1}. We define a new morphism g for all variables from $\mathcal{V} \setminus \mathcal{V}_1$ as $g(x) = f(h(x))$, where f is a function that removes all occurrences of 1 from a word. Now we will show that $g(\pi_{(2)})$ is a subword of Z_{k-1}. We see that $g(\pi_{(2)}) = f(h(\pi_{(2)}) = f(h(\pi))$ because π differs from $\pi_{(2)}$ only in variables that equal 1. So occurrence $g(\pi_{(2)})$ equals $h(\pi)$ with all 1 deleted and $g(\pi_{(2)})$ is a subword of Z_{k-1}. □

Theorem 14. *A pattern π occurs in Z_k if and only if $\mathcal{V}_1(\pi)$ is a free set and $\pi_{(2)} \to Z_{k-1}$.*

Proof. "⇒" This is a consequence of Lemma 8 and Lemma 13.
"⇐" It follows from proof of Zimin theorem and can be found in [5]. □

4 Ranked pattern-matching

It is not known (and rather unlikely true) if the pattern-matching in Zimin words is solvable in polynomial time. We introduce the following polynomially solvable version of this problem.

Compressed Ranked Pattern-Matching in Zimin Words:

Input: given a pattern π with k variables and the ranking function R
Output: a compressed instance of an occurrence of π in Z_k with the given ranking function, or information that there is no such valuation, the values of variables are given in their compressed form

The algorithm for the ranked pattern matching can be used as an auxiliary tool for pattern-matching without any ranking function given. We can just consider all *sensible* ranking functions. It gives an exponential algorithm since we do not know what the rank sequence is. Although exponential, the set of *sensible* ranking sequences can be usefully reduced due to special properties of realizable rankings.

4.1 Application of 2-SAT

In one iteration we have not only to check if the set of variables of the smallest rank i is a free set but we have to compute which of them start/end with a letter i. However in a given iteration the letter i can be treated as '1'.

In our algorithms we will use the function $\mathit{FirstLast}(\pi, W)$ which solves an instance of 2-SAT problem. It computes which variables should start-finish with the smallest rank letter, under the assumption that variables from W equal the smallest rank letter, to satisfy local properties of the Zimin word.

In the function we can treat the smallest rank letter as 1. In Zimin word there are no two adjacent '1'. This leads to the fact: for any adjacent variables xy from π either x ends with '1' or y starts with '1'. If for a given pattern we know that some variables start with '1' (or end with '1') we deduce information about successors of this variable (or predecessors). For example if we have a pattern $\beta\alpha\beta\gamma$ and we know that α starts with '1' we deduce that β does not end with '1' and then deduce that γ starts with '1'. For a given set of variables that start and end with '1' we can deduce information about all other variables in linear time (with respect to the length of the pattern).

For every variable x from the pattern we introduce two logic variables: x^{first} is true iff x starts with '1', x^{last} is true iff x ends with '1'. Now for any adjacent variables xy we create disjunctions $x^{last} \vee y^{first}$ and $\neg x^{last} \vee \neg y^{first}$. If we write the formula

$$\begin{aligned} F = &(x_1^{last} \vee x_2^{first}) \wedge (\neg x_1^{last} \vee \neg x_2^{first}) \wedge (x_2^{last} \vee x_3^{first}) \wedge (\neg x_2^{last} \vee \neg x_3^{first}) \wedge \cdots \\ &\cdots \wedge (x_{n-1}^{last} \vee x_n^{first}) \wedge (\neg x_{n-1}^{last} \vee \neg x_n^{first}) \end{aligned}$$

we have an instance of 2-SAT problem. For the variables $y_1, \ldots, y_s \in W$, that we know that valuate as '1', we expand our formula to $F \wedge y_1^{first} \wedge y_1^{last} \wedge \cdots \wedge y_l^{first} \wedge y_s^{last}$.

Example 15. If for a pattern $\beta\alpha\beta\gamma\alpha$ we know that valuation of α will be '1' we produce formula

$$\begin{gathered} (\beta^{first} \vee \alpha^{last}) \wedge (\neg\beta^{first} \vee \neg\alpha^{last}) \wedge (\alpha^{first} \vee \beta^{last}) \wedge \\ \wedge(\neg\alpha^{first} \vee \neg\beta^{last}) \wedge (\beta^{first} \vee \gamma^{last}) \wedge (\neg\beta^{first} \vee \neg\gamma^{last}) \wedge \\ \wedge(\gamma^{first} \vee \alpha^{last}) \wedge (\neg\gamma^{first} \vee \neg\alpha^{last}) \wedge (\alpha^{first}) \wedge (\alpha^{last}) \end{gathered}$$

In the general case there can be many solutions of the formula but in our example the only solution is: $\alpha^{first} = \alpha^{last} = \gamma^{first} = true$, $\beta^{first} = \beta^{last} = \gamma^{last} = false$, which means that α starts and ends with '1', β starts and ends with non-'1', γ starts with '1' and ends with non-'1'.

A positive solution to this problem is necessary for the existence of a valuation val of the variables from pattern π, such that $val(\pi) \leq Z_k$ and each variable x starts (ends) with '1' iff x^{first} is true (resp. x^{last} is true) in the solution.

It is well known that 2-SAT can be computed efficiently, consequently:

Lemma 16. *We can execute* $\mathit{FirstLast}(\pi, W)$ *in linear time.*

4.2 The algorithm: compressed and uncompressed versions

Now we present two versions of the algorithm deciding if pattern π occurs in Zimin word with given rank sequence and computing the values of variables, if there is an occurrence. First of these algorithms uses uncompressed valuations of variables and uses exponential space and second one operates on compressed valuations. If the answer is positive algorithms give valuations of variables, i.e. morphism *val* such that $val(\pi) \leq Z_k$.

Denote by $alph(\pi)$ the set of symbols (variables) in π. Let π be the pattern with given rank sequence which maximal rank equals K, $|\pi| = n, |alph(\pi)| = k$. We define the operation $\mathit{firstdel}(i,s), \mathit{lastdel}(i,s)$ of removing the first, last letter from s, respectively, if this letter is i (otherwise nothing happens), similarly define $\mathit{firstinsert}(i,s), \mathit{lastinsert}(i,s)$: inserting letter i at the beginning or at the end of s if there is no i.

In general, maximal rank can occur multiple times, but in this case, because of properties mentioned earlier in this paper, the pattern with this rank sequence is avoidable and cannot occur in Zimin word. Both algorithms assume that there is only one occurrence of the maximal rank (we denote by x_K the variable with the maximal rank).

```
**Algorithm** Uncompressed-Embedding($\pi, Z_K$)
$K :=$ maximal rank
$\mathcal{V}_i$ is the set of variables of rank $i$, for $1 \leq i \leq K$
$val(x_K) := 1;$
**for** $i = K - 1$ **downto** 1 **do**
  **for each** $x \in \mathcal{V}_i$ **do** $val(x) := 1;$
  **if** $\mathit{FirstLast}(\pi_{(i)}, \mathcal{V}_i)$ **then**
      **comment**: *we know now which variables in* $\pi_{(i)}$ *start/finish*
                *with* $i$ *due to evaluation of a corresponding 2SAT*
      **for each** $x \in \mathcal{V}_{i+1} \cup \mathcal{V}_{i+2} \cup \cdots \cup \mathcal{V}_K$ **do**
         $val(x) := \mu(val(x));$
         if $not(x^{first})$ then $val(x) := \mathit{firstdel}(1, val(x));$
         if $not(x^{last})$ then $val(x) := \mathit{lastdel}(1, val(x));$
         if $x^{first}$ then $val(x) := \mathit{firstinsert}(1, val(x));$
         if $x^{last}$ then $val(x) := \mathit{lastinsert}(1, val(x));$
  **else return** false;
```

The next algorithm is a space-efficient simulation of the previous one. We are not using the morphism μ, instead of that the values of variables are maintained in a compressed form, we are adding to the left/right decreasing sequence of integers.

```
Algorithm Compressed-Embedding($\pi, Z_K$)
$K :=$ maximal rank
$\mathcal{V}_i$ is the set of variables of rank $i$, for $1 \leq i \leq K$
$val(x_K) := K$;
for  $i = K - 1$ downto 1 do
  for each $x \in \mathcal{V}_i$ do $val(x) := k$;
  if $FirstLast(\pi_{(i)}, \mathcal{V}_i)$ then
      for each $x \in \mathcal{V}_{i+1} \cup \mathcal{V}_{i+2} \cup \cdots \cup \mathcal{V}_K$ do
          if $x^{first}$ then $val(x) := firstinsert(i, val(x))$;
          if $x^{last}$ then $val(x) := lastinsert(i, val(x))$;
  else return false;
```

Example
Below we present an example of the Uncompressed-Embedding algorithm for

$$\pi = \quad \delta\ \alpha\ \gamma\ \beta\ \lambda\ \gamma\ \alpha\ \delta\ \alpha\ \gamma\ \beta\ \alpha$$

with the rank sequence $\quad 4\ 1\ 3\ 2\ 5\ 3\ 1\ 4\ 1\ 3\ 2\ 1$

$$\overbrace{1}^{\lambda_\downarrow}$$

First we set the variable with the highest rank. $val(\lambda) = 1$

$$\overbrace{121}^{\delta_\downarrow \lambda \delta_\downarrow}$$

$i = 4$. We set $val(\delta) = 1$ and morph $val(\lambda) = 121$.
From solution of 2-SAT we know that δ starts and ends with '1',
λ starts and ends with non-'1', we set $val(\lambda) = 2$.

$$\overbrace{121}^{\delta\gamma_\downarrow}\overbrace{3}^{\lambda\gamma_\downarrow}\overbrace{121}^{\delta\gamma_\downarrow}$$

$i = 3$. We have $val(\gamma) = 1, val(\delta) = 121, val(\lambda) = 3$.
From solution of 2-SAT: γ starts and ends with '1',
δ and λ start and end with non-'1',
therefore we set $val(\lambda) = 3$, $val(\delta) = 2$

$$121\ \overbrace{3}^{\delta}\overbrace{121}^{\gamma\ \beta_\downarrow}\overbrace{4}^{\lambda}\overbrace{121}^{\gamma}\overbrace{3}^{\delta}\overbrace{121}^{\gamma\beta_\downarrow}$$

$i = 2 : val(\beta) = 1, val(\gamma) = 121, val(\delta) = 3, val(\lambda) = 4$.
From solution of 2-SAT: β starts and ends with '1', γ, δ start with '1',
end with non-'1', λ starts and ends with non='1'
Therefore $val(\gamma) = 12, val(\delta) = 13, val(\lambda) = 4$.

$$1213\ \overbrace{1214}^{\delta\,\alpha_\downarrow}\overbrace{1213}^{\gamma}\overbrace{121}^{\beta}\overbrace{5}^{\lambda}\overbrace{1213}^{\gamma\,\alpha_\downarrow}\overbrace{1214}^{\delta\,\alpha_\downarrow}\overbrace{1213}^{\gamma}\overbrace{121}^{\beta\alpha_\downarrow}$$

$i = 1 : val(\alpha) = 1, val(\beta) = 121, val(\gamma) = 1213$,
$val(\delta) = 1214, val(\lambda) = 5$.
From solution of 2-SAT: α, λ start and ends with '1',
β starts with 1, ends with non-'1', γ, δ start and end with non-'1'.

Finally we have valuation of variables, such that $val(\pi) \leq Z_5$.

$$12131\,\overbrace{214}^{\delta}\,\overbrace{1}^{\alpha}\,\overbrace{213}^{\gamma}\,\overbrace{12}^{\beta}\,\overbrace{151}^{\lambda}\,\overbrace{213}^{\gamma}\,\overbrace{1}^{\alpha}\,\overbrace{214}^{\delta}\,\overbrace{1}^{\alpha}\,\overbrace{213}^{\gamma}\,\overbrace{12}^{\beta}\,\overbrace{1}^{\alpha}$$

Example

Now we present an example of the *Compressed−Embedding* algorithm for a different pattern. Now the function *val* will be a compact representation instead of the full representation used in the last algorithm. We take:

$$\pi \;=\; \alpha\,\gamma\,\beta\,\delta\,\eta\,\alpha\,\gamma\,\beta\,\zeta\,\eta\,\alpha$$

and the ranking sequence

$$1\;3\;2\;4\;5\;1\;3\;2\;6\;5\;1$$

First we set the variable with the highest rank. $val(\zeta) = 6$

For $i = 5$ we have $\pi_{(5)} = \eta\zeta\eta$
We set $val(\eta) = 5$ and solve 2-SAT for
$$F = (\eta^{last} \vee \zeta^{first}) \wedge (\neg\eta^{last} \vee \neg\zeta^{first}) \wedge (\zeta^{last} \vee \eta^{first}) \wedge$$
$$\wedge(\neg\zeta^{last} \vee \neg\eta^{first}) \wedge (\eta^{first}) \wedge (\eta^{last}).$$
From $FirstLast(\pi_{(5)}, \eta)$ we know that ζ_{first} and ζ_{last} are false,
therefore we don't change $val(\zeta)$

$i = 4.\ \pi_{(4)} = \delta\eta\zeta\eta$
We set $val(\delta) = 4$ and execute $FirstLast(\pi_{(4)}, \delta)$. There are two solutions, we choose one of them and obtain $\eta^{first} = \eta^{last} = 0$ and $\zeta^{first} = \zeta^{last} = 1$,
therefore we change $val(\zeta) = 464$.

$i = 3 : \pi_{(3)} = \gamma\delta\eta\gamma\zeta\eta.$
We set $val(\gamma) = 3$ and execute $FirstLast(\pi_{(3)}, \gamma)$. We know that
$\delta^{first} = \delta^{last} = 0$, $\eta^{first} = 1$, $\eta^{last} = 0$ and $\zeta^{first} = \zeta^{last} = 0$,
therefore we only add 3 at the beginning of $val(\eta)$, i.e. $val(\eta) = 35$

$i = 2 : \pi_{(2)} = \gamma\beta\delta\eta\gamma\beta\zeta\eta.$
We set $val(\beta) = 2$ and execute $FirstLast(\pi_{(2)}, \beta)$. We know that
$\eta^{first} = \eta^{last} = 1$ and rest of logic variables equal 0.
We only add 2 at the beginning and end of $val(\eta)$ ($val(\eta) = 2352$)

$i = 1 : \pi_{(1)} = \pi = \alpha\gamma\beta\delta\eta\alpha\gamma\beta\zeta\eta\alpha.$
We set $val(\alpha) = 1$ and execute $FirstLast(\pi_{(1)}, \alpha)$. We know that
$\gamma^{last} = \delta^{first} = \eta^{first} = \zeta^{first} = 1$ and rest of logic variables equal 0.
We add 1 at the beginning of δ, η, ζ and at the end of γ.
We change: $val(\gamma) = 31, val(\delta) = 14, val(\eta) = 1352, val(\zeta) = 1464$.

Finally we have the compressed valuation of the variables (below we show full representations):

$$\begin{array}{cccccc}
\alpha & \beta & \gamma & \delta & \eta & \zeta \\
\boxed{1} & \boxed{2} & \boxed{3\,1} & \boxed{1\,4} & \boxed{1\,3\,5\,2} & \boxed{1\,4\,6\,4} \\
1 & 2 & 3\,1 & 1\,4 & 13121512 & 141213121612131214
\end{array}$$

Our algorithms rely on the following lemma.

Lemma 17. *Let $i \in \{1, \ldots, k\}$ and the pattern $\pi_{(i+1)}$ occurs in Z_{k-i}. Pattern $\pi_{(i)}$ (equal $\pi_{(i+1)}$ with additional variables from $\mathcal{V}_i$) occurs in Z_{k-i+1} iff the corresponding 2-SAT problem has a solution. Moreover every immersion of $\pi_{(i)}$, such that valuations of variables from $\mathcal{V}_i$ equal '1', corresponds to the solution of 2-SAT problem, such that valuation of every variable satisfies logic constraints on first and last character.*

Proof.
" $\Leftarrow$ "
First we observe that solution of 2-SAT problem guarantees that $\mathcal{V}_i$ is a free set. We put $A = \{v \in \pi_{(i)} : v^{last} = 0\}$ and $B = \{v \in \pi_{(i)} : v^{first} = 1\}$, which satisfy Definition 9. From Theorem 14 $\pi_{(i)}$ occurs in Z_{k-i+1}. Now we use Zimin morphism μ on variables from $\pi_{(i+1)}$ and set every variable from $\mathcal{V}_i$ to '1', then we modify (adding or removing '1' at the beginning or end) every valuation accordingly to logic constraints from the 2-SAT solution. Similarly as in the Zimin theorem proof ([5]) we see that properties of A and B guarantee that modified valuations concatenate into proper immersion in Z_{k-i+1}.
" $\Rightarrow$ "
We have immersion of $\pi_{(i)}$ in Z_{k-i+1} with valuation $val(v)$, such that for $v \in \mathcal{V}_i$ $val(v) = 1$. We give a solution to 2-SAT problem as follows: for every variable v from $\pi_{(i)}$ we set $v^{first} = 1$ iff $val(v)$ starts with '1', $v^{last} = 1$ iff $val(v)$ ends with '1'. Because Zimin word is 1-interleaved this solution is correct. □

Theorem 18. *The compressed ranked pattern matching in Zimin words can be solved in time $O(nk)$ and (simultaneously) space $O(n+k^2)$, where n is the size of the pattern and k is the highest rank of a variable. A compressed instance of the pattern can be constructed within the same complexities, if there is any solution.*

Proof. We use the *Compressed − Embedding* algorithm. First we embed $\pi_{(k)}$ into Z_1. Then we check for a subsequent i, $k-1 \geq i \geq 1$, if it is possible to embed $\pi_{(i)}$ having embedding of $\pi_{(i+1)}$ using suitable 2-SAT and Lemma 17. Non-existence of immersion of $\pi_{(i)}$ implies that whole π does not occur into Z_k. Otherwise we get compressed valuation of variables, such that $val(\pi) \leq Z_k$.

We will consider time complexity of the Compressed-Embedding algorithm. Because $|V_1| \cup |V_2| \cup \cdots \cup |V_k| = k$ first **for each** loop executes exactly k times during whole execution.

We solve 2-SAT problem exactly $k-1$ times for $\pi_{(i)}$ (length of formula is linear with respect to $|\pi_{(i)}|$) and for every i $|\pi_{(i)}| \leq |\pi| = n$. That gives complexity $O(nk)$ for this step.

We execute second **for each** loop $k-1$ times. In each iteration we have:

$$|\mathcal{V}_{i+1} \cup \mathcal{V}_{i+2} \cup \cdots \cup \mathcal{V}_K| \leq |\mathcal{V}_1 \cup \mathcal{V}_2 \cup \cdots \cup \mathcal{V}_K| = k$$

Hence complexity for this step is $O(k^2)$.
Finally, the algorithm has time complexity $O(k + nk + k^2) = O(nk)$, because $k \leq n$.

We have some choice when applying 2-SAT, since many satisfying valuation are sometimes possible. By slightly modifying the application of 2-SAT we can obtain the following result.

Theorem 19. *For a given ranked pattern the compressed shortest instance and lexicographically smallest instance of the ranked pattern occurring in Z_k can be constructed in time $O(nk)$ and (simultaneously) space $O(n+k^2)$ (if there is any instance).*

References

1. D. R. Bean, A. Ehrenfeucht, and G. F. McNulty: *Avoidable patterns in strings of symbols*, Pacific J. Math, 1979.
2. C. E. Heitsch: *Generalized Pattern Matching and the Complexity of Unavoidability Testing*, Lecture Notes in Computer Science, 2001.
3. M. Lothaire: *Algebraic Combinatorics on Words*, Cambridge University Press, 2002.
4. A. Thue: *Über unendliche Zeichenreihen*, Norske Vid. Selsk. Skr. I. Mat. Nat. Kl. Christiana 7, 1906.
5. A. I. Zimin: *Blocking sets of terms*, Mat. Sb. (N.S.) 119(161), 1982.