Efficient Indexes for Jumbled Pattern Matching with Constant-Sized Alphabet

Tomasz Kociumaka[1], Jakub Radoszewski[1], and Wojciech Rytter[1,2⋆]

[1] Faculty of Mathematics, Informatics and Mechanics,
University of Warsaw, Warsaw, Poland
[kociumaka,jrad,rytter]@mimuw.edu.pl
[2] Faculty of Mathematics and Computer Science,
Copernicus University, Toruń, Poland

Abstract. We introduce efficient data structures for an indexing problem in non-standard stringology — jumbled pattern matching. Moosa and Rahman [J. Discr. Alg., 2012] gave an index for jumbled pattern matching for the case of binary alphabets with $O(\frac{n^2}{\log^2 n})$-time construction. They posed as an open problem an efficient solution for larger alphabets. In this paper we provide an index for any constant-sized alphabet. We obtain the first $o(n^2)$-space construction of an index with $o(n)$ query time. It can be built in $O(n^2)$ time. Precisely, our data structure can be implemented with $O(n^{2-\delta})$ space and $O(m^{(2\sigma-1)\delta})$ query time for any $\delta > 0$, where m is the length of the pattern and σ is the alphabet size ($\sigma = O(1)$). We also break the barrier of quadratic construction time for non-binary constant alphabet simultaneously obtaining poly-logarithmic query time.

1 Introduction

The problem of *jumbled pattern matching* is a variant of the standard pattern matching problem. The *match* between a given pattern and a factor of the word is defined in a nonstandard way. In this paper by a *match* of two words we mean their commutative (Abelian) equivalence: one word can be obtained from the other by permuting its symbols. This relation can be conveniently described using *Parikh vectors*, which show frequency of each symbol of the alphabet in a word: u and v are commutatively equivalent (denoted as $u \approx v$) if and only if their Parikh vectors are equal. In the jumbled pattern matching the query pattern is given as a Parikh vector, which in our case is of a constant size (due to small alphabet).

Several results related to indexes for jumbled pattern matching in binary words have been obtained recently. Cicalese et al. [8] proposed an index with $O(n)$ size and $O(1)$ query time and gave an $O(n^2)$ time construction algorithm. The key observation used in this index is that if a word contains two factors of length ℓ containing i and j ones, $i < j$, respectively, then it must contain a factor

⋆ Supported by grant no. N206 566740 of the National Science Centre.

of length ℓ with any intermediate number of ones. The construction time was improved independently by Burcsi et al. [4] (see also [5, 6]) and Moosa and Rahman [20] to $O(n^2/\log n)$ and then by Moosa and Rahman [21] to $O(n^2/\log^2 n)$. An index for trees vertex-labeled with $\{0, 1\}$ achieving the same complexity bounds ($O(n^2/\log^2 n)$ construction time, $O(n)$ size and $O(1)$ query time) was given in [17]. The general problem of computing an index for jumbled pattern matching in trees and graphs is known to be NP-complete [13, 19].

Moosa and Rahman [20, 21] posed an open problem for a construction of an $o(n^2)$ indexing scheme for general alphabet (with $o(n)$ query time). In particular, even for a ternary alphabet none was known, the basic observation used to obtain a binary index does not hold for any larger alphabet. We prove that the answer for this problem is positive for any constant-sized alphabet. We show an $O(n^2(\log\log n)^2/\log n)$ time and space construction of an index that enables queries in $O((\frac{\log n}{\log\log n})^{2\sigma-1})$ time. We also give a solution with $O(n^{2-\delta})$ size of the index, $O(n^2)$ construction time and $O(m^{(2\sigma-1)\delta})$ query time. Here σ is the size of the alphabet and m is the length of the pattern, that is, the sum of components of the Parikh vector of the pattern. Our construction algorithms are randomized (Las Vegas) and work in word-RAM model with word size $\Omega(\log n)$. The latter index is described in Section 3 and in Section 4 (improvement from $n^{(2\sigma-1)\delta}$ to $m^{(2\sigma-1)\delta}$ in query time) and the former index is given in Section 5 (auxiliary tools) and in Section 6.

A notion closely related to jumbled pattern matching are Abelian periods, first defined in [9]. The pair (i, p) is an Abelian period of w if $w = u_0u_1 \dots u_ku_{k+1}$ where $u_1 \approx u_2 \approx \dots \approx u_k$, u_0 and u_{k+1} contain a subset of letters of u_1, and $|u_0| = i$, $|u_1| = p$. Recently there have been a number of results related to efficient algorithms for Abelian periods [9, 14, 15, 18, 10]. There have also been several combinatorial results on Abelian complexity in words [1, 7, 11, 12] and partial words [2, 3].

2 Preliminaries

In this paper we assume that the alphabet Σ is $\{1, 2, \dots, \sigma\}$ for $\sigma = O(1)$. For a word $w \in \Sigma^n$ by $w[i\,..\,j]$ denote a *factor* of w equal to $w_i \dots w_j$. We say that the factor $w[i\,..\,j]$ *occurs* at the position i.

Let $\#_s(x)$ denote the number of occurrences of the letter s in x. The Parikh vector $\Psi(x)$ of the word $x \in \Sigma^*$ is defined as:

$$\Psi(x) \;=\; (\#_1(x),\, \#_2(x),\, \#_3(x),\, \dots,\, \#_\sigma(x)).$$

We say that the words x and y are *commutatively equivalent* (denoted as $x \approx y$) if y can be obtained from x by a permutation of its letters. Observe that we have:

$$x \approx y \iff \Psi(x) = \Psi(y).$$

Example 1. $1221 \approx 2211$, since $\Psi(1221) = (2, 2) = \Psi(2211)$.

We define $Occ(p)$ as the set of all positions where factors of w with the Parikh vector p occur. If $Occ(p) \neq \emptyset$, we say that p *occurs* in w.

The problem of *indexing for jumbled pattern matching* (also called indexing for permutation matching [20, 21]) is defined as follows:

Preprocessing: build an index for a given word w of length n;

Query: for a given Parikh vector (a pattern) p decide whether p occurs in w.

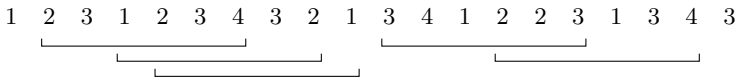

Fig. 1. Let $w = 1\,2\,3\,1\,2\,3\,4\,3\,2\,1\,3\,4\,1\,2\,2\,3\,1\,3\,4\,3$. The factor $2\,3\,1\,2\,3\,4$ occurs (as a word) starting at position 2. We have $\Psi(2\,3\,1\,2\,3\,4) = (1, 2, 2, 1)$, hence the jumbled pattern $p = (1, 2, 2, 1)$ (Parikh vector) also occurs at position 2. There are several occurrences of the jumbled pattern p: $Occ(p) = \{2, 4, 5, 11, 14\}$.

Define the *norm* of a Parikh vector $p = (p_1, p_2, \ldots, p_\sigma)$ as:

$$|p| = \sum_{i=1}^{\sigma} |p_i|.$$

For two Parikh vectors p, q, by $p + q$ and $p - q$ we denote their component-wise sum and difference. We define the *extension* sets of Parikh vectors:

$$Ext_{<r}(p) = \{p + p' : |p'| < r\}, \quad Ext_r(p) = \{p + p' : |p'| = r\}.$$

Also define

$$Ext'_{<r}(p) = Ext_{<r}(p) \cap \{p' : Occ(p) \cap Occ(p') \neq \emptyset\}.$$

For a set X of Parikh vectors, we define:

$$Ext_{<r}(X) = \bigcup_{p \in X} Ext_{<r}(p), \quad Ext'_{<r}(X) = \bigcup_{p \in X} Ext'_{<r}(p).$$

Lemma 2. *For any Parikh vector p and integer $r \geq 0$, $|Ext_r(p)| = O(r^{\sigma-1})$, $|Ext_{<r}(p)| = O(r^\sigma)$ and $|\{q : p \in Ext_r(q)\}| = O(r^{\sigma-1})$.*

Proof. Both $|Ext_r(p)|$ and $|\{q : p \in Ext_r(q)\}|$ are bounded by the number of Parikh vectors of norm r, which is $\binom{r+\sigma-1}{\sigma-1}$, since each Parikh vector corresponds to a placement of r indistinguishable balls ('positions') into σ distinguishable urns ('letters').

To bound $|Ext_{<r}(p)|$ it suffices to observe that

$$Ext_{<r}(p) = \bigcup_{k=0}^{r-1} Ext_k(p).$$

□

Let us also introduce an efficient tool for determining Parikh vectors of factors of a given word.

Lemma 3. *After $O(n)$ time preprocessing, the Parikh vector $\Psi(w[i\,..\,j])$ for any $1 \leq i \leq j \leq n$ can be computed in $O(1)$ time.*

Proof. For each $k \in \{0, \ldots, n\}$ we precompute $\Psi(w[1\,..\,k])$ in $O(n)$ time. Then

$$\Psi(w[i\,..\,j]) \;=\; \Psi(w[1\,..\,j]) - \Psi(w[1\,..\,i-1]). \qquad \square$$

3 Index with Sublinear Time Queries

A Parikh vector which occurs in w is called an *Abelian factor* of w. Observe that the *zero vector* is an Abelian factor of every word, since it corresponds to the empty word.

Define $Abelian(w)$ as the set of all Abelian factors of w. For example:

$$Abelian(a^k b^k) \;=\; \{(i,j) \;:\; 0 \leq i,j \leq k\}.$$

For a positive integer L we say that a pair $(\mathcal{A}, \mathcal{B})$ of two disjoint subsets of $Abelian(w)$ is *L-good* if the following conditions are satisfied:

(1) $\sum_{p \in \mathcal{A} \cup \mathcal{B}} |Occ(p)| \;\leq\; n^2/L$;

(2) $|Ext_{<L}(\mathcal{B})| = O(n^2/L)$;

(3) $|Occ(p)| \leq L^\sigma$ for each $p \in \mathcal{A}$;

(4) for each $z \in Abelian(w)$ we have:

$$z \in Ext'_{<L}(\mathcal{B}) \;\;\text{ or }\;\; \exists_{p \in \mathcal{A},\, 0 \leq |z|-|p| < L} \; Occ(z) \cap Occ(p) \neq \emptyset.$$

Note that condition (4) could also be stated as $z \in Ext'_{<L}(\mathcal{B}) \cup Ext_{<L}(\mathcal{A})$, however we choose the above statement due to operational reasons (see the following Query algorithm).

Let

$$\mathcal{F}_L \;=\; \{p \in Abelian(w) \;:\; |p| \bmod L = 0\}.$$

Elements of $\mathcal{F}_L$ are called L-factors, clearly $|\mathcal{F}_L| \leq \frac{n^2}{L}$.

We partition the set of L-factors into the set $\mathcal{L}_L$ of *light* Abelian factors (small number of occurrences) and the set $\mathcal{H}_L$ of *heavy* Abelian factors (more occurrences in w). More formally:

$$\mathcal{L}_L = \{p \in \mathcal{F}_L : |Occ(p)| \leq L^\sigma\},$$
$$\mathcal{H}_L = \{p \in \mathcal{F}_L : |Occ(p)| > L^\sigma\} \cup \{\bar{0}\}.$$

Lemma 4. *The pair* $(\mathcal{L}_L, \mathcal{H}_L)$ *is* L-good.

Proof. The total size of $Occ(p)$ for all p, such that $|p| = k \cdot L$ for a fixed k, is at most n. Hence, the total size of $Occ(p)$ for all $p \in \mathcal{F}_L$ is at most $\frac{n^2}{L}$, which gives part (1) of the definition of an L-good pair.

Part (3) follows from the definition of $\mathcal{L}_L$. As for part (2), by Lemma 2, for each $p \in \mathcal{H}_L$ we have

$$|Ext_{<L}(p)| = O(L^\sigma) = O(|Occ(p)|).$$

The latter inequality follows from the definition of $\mathcal{H}_L$. This shows that the size of $Ext_{<L}(\mathcal{H}_L)$ is bounded by the total size of $Occ(p)$ for $p \in \mathcal{H}_L$, which we have already shown (part (1)) to be bounded by $\frac{n^2}{L}$.

As for property (4), consider any $z \in Abelian(w)$ and its occurrence at position i. Let $p = \Psi(w[i\,..\,i + k \cdot L - 1])$, where $k \cdot L \leq |z| < (k+1) \cdot L$. Then either $p \in \mathcal{H}_L$ and therefore $z \in Ext'_{<L}(p)$ or $p \in \mathcal{L}_L$ and clearly $i \in Occ(p)$. □

Data Structure. The index consists of the following parts:

1. the sets $\mathcal{L}_L$, $\mathcal{H}_L$ and $Ext'_{<L}(\mathcal{H}_L)$;
2. $Occ(p)$ for each $p \in \mathcal{L}_L$.

All the components are stored in hash tables indexed by p. With perfect hashing [16] we obtain $O(1)$ access time and $O(n^2/L)$ space. The following algorithm realizes a query. Note that the only Abelian factor of length 0 is heavy. Hence, if $|z| < L$ then z is an Abelian factor of w if and only if $z \in Ext'_{<L}(\mathcal{H}_L)$.

```
Algorithm QUERY(z)
    if z ∈ Ext'_{<L}(H_L) then
        return true;
    r := |z| mod L;
    foreach p : z ∈ Ext_r(p) do
        if p ∈ L_L then
            foreach i ∈ Occ(p) do
                if w[i..i + |z| − 1] ≈ z then
                    return true;
    return false;
```

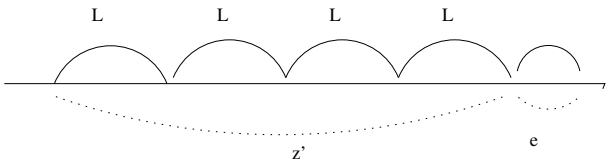

Fig. 2. When searching for the Abelian factor z we look for any L-factor z' and (short) factor e such that $z = z' + e$ and $|e| < L$.

In the query algorithm we check if $z \in Ext'_{<L}(\mathcal{H}_L)$ or, otherwise, if there exists $p \in \mathcal{L}_L$, $0 \le |z| - |p| < L$, such that z occurs as an extension of p. Hence, the correctness of the query algorithm follows from property (4) of an L-good pair. Let us analyze the complexity of the data structure.

Theorem 5. *For any integer $L > 0$ there exists an index for jumbled pattern matching with $O(n^2/L)$ space and $O(L^{2\sigma-1})$ query time. The preprocessing time is $O(n^2)$.*

Proof. Assume that the elements of $\mathcal{F}_L$ are indexed using a hash table. Let us consider the complexity of the main iteration of a single query. By Lemma 2, $|\{p : p \in Ext_r(z)\}| \le L^{\sigma-1}$, and for any $p \in \mathcal{L}_L$, by definition, $|Occ(p)| \le L^\sigma$. Thus, using constant-time equivalence queries from Lemma 3, we obtain the desired $O(L^{2\sigma-1})$ query time.

The index size is bounded by

$$|\mathcal{F}_L| + \sum_{p \in \mathcal{F}_L} |Occ(p)| + |Ext'_{<L}(\mathcal{H}_L)|$$

which is $O(n^2/L)$ by the conditions (1-2) of an L-good pair.

Finally consider the preprocessing time. The sets $\mathcal{F}_L$, $\mathcal{L}_L$ and $\mathcal{H}_L$, and also $Occ(p)$ for all $p \in \mathcal{F}_L$, can be computed in $O(n^2/L)$ time. To compute $Ext'_{<L}(\mathcal{H}_L)$, we consider each $z \in \mathcal{H}_L$, all the elements of $Occ(z)$ and all the extensions $z + e$ of the corresponding occurrences of z by at most L letters. This yields $O(n^2/L \cdot L) = O(n^2)$ time. □

Corollary 6. *For any $0 < \delta < 1$ there exists an index for jumbled pattern matching with $O(n^{2-\delta})$ space and $O(n^{(2\sigma-1)\delta})$ query time.*

Proof. We take $L = n^\delta$ and apply Theorem 5. □

4 The Case of Small Patterns

While $O(n^{(2\sigma-1)\delta})$ is sublinear in n for small δ, it is still rather large, and, especially for very small patterns, might be considered unsatisfactory. We modify the data structure to handle such patterns much more efficiently, in $O(m^{(2\sigma-1)\delta})$ time for patterns of length m. We start with an auxiliary data structure.

Lemma 7. *For any $0 < \delta < 1$ there exists an index for jumbled pattern matching with $O(n \cdot k^{1-\delta})$ space and $O(k^{(2\sigma-1)\delta})$ query time for patterns of size that is at most k.*

Proof. We slightly change the definition of L-factors, we only take the L-factors of size at most k. Let $\mathcal{F}_{L,k}$ denote the set of these factors. Similarly as in the case of $\mathcal{F}_L$ we obtain $|\mathcal{F}_{L,k}| \leq \frac{n \cdot k}{L}$.

Now we take $L = k^\delta$ and the rest of the construction is essentially the same as before. The size of the data structure is $O(\frac{n \cdot k}{L})$, which is $O(n \cdot k^{1-\delta})$. The query time is $O(L^{2\sigma-1})$, hence of order $O(k^{(2\sigma-1)\delta})$. □

Theorem 8. *For any $\delta > 0$ there exists an index for jumbled pattern matching with $O(n^{2-\delta})$ space and $O(m^{(2\sigma-1)\delta})$ query time, where m is the size of the pattern.*

Proof. Let

$$K \;=\; \{2^i \;:\; 0 \leq i \leq \lfloor \log n \rfloor\} \cup \{n\}.$$

We can precompute the data structures from Lemma 7 for each $k \in K$. The total size will be of order:

$$n^{2-\delta} + \sum_{i=0}^{\lfloor \log n \rfloor} n \cdot 2^{i(1-\delta)} \;=\; n^{2-\delta} + n \cdot O(2^{(1-\delta)\log n}) = O(n^{2-\delta}).$$

To answer a query about a pattern p of size m we take

$$k \;=\; \min\{j \in K \;:\; j \geq m\}.$$

Then we apply the query algorithm from Lemma 7 (using only the part of the data structure relevant to k). This completes the proof. □

In particular if we take $\delta = 1/((2\sigma-1)a)$ then we have a more concrete result.

Observation 9. *For any integer $a > 1$ there exists an index for jumbled pattern matching with $O(n^{2-1/((2\sigma-1)a)})$ space and $O(\sqrt[a]{m})$ query time.*

5 Efficient Merging of Packed Sets

In this section by merging we mean computing a set-theoretic union, i.e. at most one copy of each element is preserved. We merge large families of sets whose union is relatively small. We aim at sublinear time in the total size of those families, which requires suitable compact encoding of sets. The algorithm that we develop in this section is used to obtain an $o(n^2)$ time construction algorithm for an index for jumbled pattern matching, which is shown in the following section.

Let $U = \{1, 2, \ldots, N\}$, where $N = \left\lceil \left(\frac{\log n}{\log \log n} \right)^\sigma \right\rceil$ and $M(n) = \delta \cdot \frac{\log n}{(\sigma \log \log n)}$.

The set U is called the *universe*, and its subsets of size not larger than $M(n)$ are called here *small sets*. We have $\log |U| = \sigma \log \log n$ and $|U|^{M(n)} = 2^{\log |U| \cdot M(n)} = n^\delta$. Consequently, we obtain a bound for the number of small sets.

Observation 10. *There are $O(n^\delta)$ small sets.*

Assume all small subsets are listed in lexicographic order ($t = O(n^\delta)$):

$$\mathcal{S} = \{S_0, S_1, \ldots, S_t\}.$$

Then each small subset can be identified by its rank in the list above. A set of identifiers $\mathcal{X} = \{\gamma_1, \gamma_2, \ldots, \gamma_m\} \subseteq \{0, 1, \ldots, t\}$ represents a family $\mathcal{R} = S_{\gamma_1}, S_{\gamma_2}, \ldots, S_{\gamma_m}$. We say that each identifier in $\mathcal{X}$ is a *packed* representation of the subset of U and $\mathcal{X}$ is the *packed* version of a family $\mathcal{R}$ of small sets. We denote:

$$\mathcal{R} = \mathit{UNPACK}(\mathcal{X}) \text{ and } \mathcal{X} = \mathit{PACK}(\mathcal{R}).$$

For a family $\mathcal{R}$ of subsets of U denote by $\mathit{Merge}(\mathcal{R})$ the sorted set of all elements of $\bigcup_{S\in\mathcal{R}} S$, with duplicates removed (each element has unique occurrence in the merge). Denote

$$\mathit{PackedMerge}(\mathcal{X}) = \mathit{Merge}(\mathit{UNPACK}(\mathcal{X})).$$

Define the following **Packed Merging Problem:**

Input: a family $\mathcal{X} = \{\gamma_1, \gamma_2, \ldots, \gamma_m\}$ of integers (packed small sets);

Output: $\mathit{PackedMerge}(\mathcal{X})$.

Example 11. Let $U = \{1, 2, 3, 4\}$ and $M(n) = 2$. Then:
$\mathit{PackedMerge}(\{2, 4, 7\}) = \mathit{Merge}(S_2, S_4, S_7) = \{1, 2, 4\}$.
In this case the lexicographically ordered ordered list of small sets is:
$S_0 = \emptyset$, $S_1 = \{1\}$, $S_2 = \{1, 2\}$, $S_3 = \{1, 3\}$, $S_4 = \{1, 4\}$, $S_5 = \{2\}$,
$S_6 = \{2, 3\}$, $S_7 = \{2, 4\}$, $S_8 = \{3\}$, $S_9 = \{3, 4\}$, $S_{10} = \{4\}$.

Lemma 12. [PackedMerge Lemma] *Let $\delta = \frac{1}{2} - \varepsilon$ for any $0 < \varepsilon < \frac{1}{2}$. Then after $O(n)$ time preprocessing, for each packed family $\mathcal{X}$ of m integers $S = \mathit{PackedMerge}(\mathcal{X})$ can be computed in $O(|S| + (m + \log^\sigma n) \log\log n)$ time.*

Proof. For a set S of integers we write $S \leq K$ if all elements of S are smaller than or equal to K, similarly we write $S > K$ if all the elements are greater than K. For two identifiers i, j denote $\mathit{SmallSplit}(i, j, K) = (p, q)$, where (p, q) is any pair of indices of subsets such that:

$$S_p \cup S_q = S_i \cup S_j \text{ and } (S_p \leq K \text{ or } S_p > K).$$

For an identifier i and an integer K also denote $\mathit{Split}(i, K) = (p, q)$, where (p, q) is any pair of indices of subsets such that:

$$S_p \cup S_q = S_i \text{ and } S_p \leq K \text{ and } S_q > K.$$

Note that the number of triples (i, j, K) is $o(n)$. Hence:

Claim. After $o(n)$-time preprocessing each *SmallSplit* and *Split* query can be answered in constant time.

Using *SmallSplit* and *Split* operations we can *scan* the sequence $\mathcal{X}$, then each time we process two current consecutive sets S_i, S_{i+1}, S_{i+1} is replaced by S_q and we have $S_p \leq K$ or $S_p > K$, where $SmallSplit(i, j, K) = (p, q)$.

Algorithm LargeSplit($\mathcal{X}, K$)
  Assume $\mathcal{X} = \{\gamma_1, \gamma_2, \ldots, \gamma_m\}$;
  $\mathcal{X}_1 := \mathcal{X}_2 := \emptyset$;
  for $i := 1$ **to** $m - 1$ **do**
    $(p, q) := SmallSplit(\gamma_i, \gamma_{i+1}, K)$;
    $\gamma_{i+1} := q$;
    if $S_p \leq K$ **then** add p to $\mathcal{X}_1$ **else** add p to $\mathcal{X}_2$;
  $(p, q) := Split(\gamma_m, K)$;
  add p to $\mathcal{X}_1$;
  add q to $\mathcal{X}_2$;
  return ($\mathcal{X}_1$, $\mathcal{X}_2$);

In this way we have shown:

Claim. We can compute in $O(|\mathcal{X}|)$ time two families $\mathcal{X}_1$, $\mathcal{X}_2$ of packed sets such that:

- $PackedMerge(\mathcal{X}_1) \cup PackedMerge(\mathcal{X}_2) = PackedMerge(\mathcal{X})$;
- $PackedMerge(\mathcal{X}_1) \leq K$, $PackedMerge(\mathcal{X}_2) > K$.
- The total number of packed sets in $\mathcal{X}_1$, $\mathcal{X}_2$ is at most $|\mathcal{X}| + 1$.

After at most $\log |U|$ operations *LargeSplit*, each time applied to a smaller range of integers in U, we arrive at the situation when $\mathcal{X}$ is transformed into a series of nonempty packed families, each of them contains packed subsets included in the subrange of U of size at most $M(n)$.

Algorithm Generate($\mathcal{X}$)
  $Queue := \{(\mathcal{X}, [0, N])\}$; $OutputList := \emptyset$;
  while $Queue \neq \emptyset$ **do**
    $(\mathcal{X}', \Delta) := delete(Queue)$; $middle := mid(\Delta)$;
    $(\mathcal{X}_1, \mathcal{X}_2) :=$ LargeSplit($\mathcal{X}', middle$);
    if $|\Delta|/2 \leq M(n)$ **then** add $\mathcal{X}_1$, $\mathcal{X}_2$ to $OutputList$;
    else add ($\mathcal{X}_1$, $left(\Delta)$), ($\mathcal{X}_2$, $right(\Delta)$) to $Queue$;
  return $OutputList$;

The algorithm above returns the set of families of packed subsets, for each family all its sets are subsets of the same interval of size $M(n)$. For an interval

Δ let $mid(\Delta)$ be the middle point in Δ and $left(\Delta)$, ($right(\Delta)$) be the left (resp. right) half of Δ.
Similarly as for *SmallSplit* queries, we obtain the following claim that enables us to efficiently merge packed sets belonging to the same short range.

Claim. After $O(n)$ time preprocessing for each two packed sets S_i, S_j which are subsets of some range Δ', $|\Delta'| \leq M(n)$, we can compute the packed version of their union (the identifier p such that $S_p = S_i \cup S_j$) in $O(1)$ time.

Now we can compute union of each subfamily in *OutputList* in time proportional to the number of returned elements. Since returned sets are disjoint for different subfamilies the time is proportional to the total number of returned elements.

The total time of all operations *LargeSplit* is $O((m+|U|)\log|U|) = O((m+\log^\sigma n)\log\log n)$ since we perform operations on $O(\log|U|)$ *levels* (a level corresponds iterations with the same $|\Delta|$) and at each level we spend $O(m+|U|)$ time. □

6 Reducing Preprocessing Time

In Section 3 we have presented a subquadratic-space and sublinear query-time index. However the construction time was quadratic. Now we give an index which can be built slightly faster. The following theorem solves an open problem stated by Moosa and Rahman [20, 21] for the case of constant-sized alphabet.

Theorem 13. *Each query in the index for jumbled pattern matching can be answered in* $O((\frac{\log n}{\log\log n})^{2\sigma-1})$ *time after preprocessing in* $O\left(\frac{(\log\log n)^2}{\log n}\cdot n^2\right)$ *time and space.*

Proof. The queries work as in Theorem 5 with $L = \lfloor M(n)\rfloor$. Recall that $M(n) = \delta\cdot\log n/(\sigma\log\log n)$, where $\delta = \frac{1}{2}-\varepsilon$ for any $0 < \varepsilon < \frac{1}{2}$. For simplicity we extend the word w with L trailing sentinel letters. The index itself is also the same, its space complexity is $O(n^2/L) = O\left(\frac{\log\log n}{\log n}\cdot n^2\right)$. As described in the proof of Theorem 5, all the parts of the preprocessing excluding the computation of $Ext'_{<L}(\mathcal{H}_L)$ work in $O(n^2/L)$ time. The missing component is constructed by a reduction to Packed Merging Problem.

Let N be the number of distinct Abelian factors e of the input word such that $|e| < L$. Then $N \leq L^\sigma = O(\log^\sigma n)$. All such e's can be computed and assigned different identifiers from $\{0, 1, \ldots, N\}$ in $O(nL)$ time. These identifiers form the universe in the Packed Merging Problem.

Denote by A the number of distinct (ordinary) factors u of the input word such that $|u| = L$. We have $A \leq \sigma^L = O(n^\delta)$. All such factors can be computed and assigned different identifiers from $\{0, 1, \ldots, A\}$ in $O(n\cdot(L+\log n)) = O(n\log n)$ time.

For each factor $u \in \{0, 1, \ldots, A\}$ we can also compute in $O(n^\delta\cdot L)$ total time the set $S(u)$ of identifiers of all Parikh vectors corresponding to prefixes of u.

Note that the size of $S(u)$ is $L = \lfloor M(n) \rfloor$. Hence, the sets $S(u)$ form the *small sets* from the PackedMerge problem.

Now the extension sets $Ext'_{<L}(p)$ for each $p \in \mathcal{H}_L$ are computed separately. We consider all positions in $Occ(p)$, for each such position i we take the identifier u of the L-letter word coming after the corresponding occurrence of p. To compute $Ext'_{<L}(p)$, it suffices to find all the distinct elements among all the sets $S(u)$ and add p to each of them. By the PackedMerge Lemma, this can be performed in $O(|S| + (\log n)^\sigma \log\log n + |Occ(p)| \cdot \log\log n)$ time, where $S = Ext'_{<L}(p)$. In total $|S| > (\log n)^\sigma$ both sum up to at most $O(n^2/L)$ and $|Occ(p)| \cdot \log\log n$ sum up to $O(n^2 \log\log n/L)$, which yields the time complexity of the construction. □

Acknowledgement

The authors would like to thank several researchers present at the Stringmasters 2013 workshop in Verona for introducing the problem and comments on the preliminary solution: Péter Burcsi, Ferdinando Cicalese, Gabriele Fici, Travis Gagie, Arnaud Lefebvre and Zsuzsanna Lipták. We are especially grateful to Ferdinando Cicalese and Travis Gagie for very valuable remarks.

References

1. S. V. Avgustinovich, A. Glen, B. V. Halldórsson, and S. Kitaev. On shortest crucial words avoiding Abelian powers. *Discrete Applied Mathematics*, 158(6):605–607, 2010.
2. F. Blanchet-Sadri, J. I. Kim, R. Mercas, W. Severa, and S. Simmons. Abelian square-free partial words. In A. H. Dediu, H. Fernau, and C. Martín-Vide, editors, *LATA*, volume 6031 of *Lecture Notes in Computer Science*, pages 94–105. Springer, 2010.
3. F. Blanchet-Sadri and S. Simmons. Avoiding Abelian powers in partial words. In G. Mauri and A. Leporati, editors, *Developments in Language Theory*, volume 6795 of *Lecture Notes in Computer Science*, pages 70–81. Springer, 2011.
4. P. Burcsi, F. Cicalese, G. Fici, and Z. Lipták. On table arrangements, scrabble freaks, and jumbled pattern matching. In P. Boldi and L. Gargano, editors, *FUN*, volume 6099 of *Lecture Notes in Computer Science*, pages 89–101. Springer, 2010.
5. P. Burcsi, F. Cicalese, G. Fici, and Z. Lipták. Algorithms for jumbled pattern matching in strings. *Int. J. Found. Comput. Sci.*, 23(2):357–374, 2012.
6. P. Burcsi, F. Cicalese, G. Fici, and Z. Lipták. On approximate jumbled pattern matching in strings. *Theory Comput. Syst.*, 50(1):35–51, 2012.
7. J. Cassaigne, G. Richomme, K. Saari, and L. Q. Zamboni. Avoiding Abelian powers in binary words with bounded Abelian complexity. *Int. J. Found. Comput. Sci.*, 22(4):905–920, 2011.
8. F. Cicalese, G. Fici, and Z. Lipták. Searching for jumbled patterns in strings. In J. Holub and J. Žďárek, editors, *Proceedings of the Prague Stringology Conference 2009*, pages 105–117, Czech Technical University in Prague, Czech Republic, 2009.
9. S. Constantinescu and L. Ilie. Fine and Wilf's theorem for Abelian periods. *Bulletin of the EATCS*, 89:167–170, 2006.

10. M. Crochemore, C. Iliopoulos, T. Kociumaka, M. Kubica, J. Pachocki, J. Radoszewski, W. Rytter, W. Tyczyński, and T. Waleń. A note on efficient computation of all Abelian periods in a string. *Information Processing Letters*, 113(3):74–77, 2013.
11. J. D. Currie and A. Aberkane. A cyclic binary morphism avoiding Abelian fourth powers. *Theor. Comput. Sci.*, 410(1):44–52, 2009.
12. J. D. Currie and T. I. Visentin. Long binary patterns are Abelian 2-avoidable. *Theor. Comput. Sci.*, 409(3):432–437, 2008.
13. M. R. Fellows, G. Fertin, D. Hermelin, and S. Vialette. Upper and lower bounds for finding connected motifs in vertex-colored graphs. *J. Comput. Syst. Sci.*, 77(4):799–811, 2011.
14. G. Fici, T. Lecroq, A. Lefebvre, and É. Prieur-Gaston. Computing Abelian periods in words. In J. Holub and J. Žďárek, editors, *Proceedings of the Prague Stringology Conference 2011*, pages 184–196, Czech Technical University in Prague, Czech Republic, 2011.
15. G. Fici, T. Lecroq, A. Lefebvre, E. Prieur-Gaston, and W. Smyth. Quasi-linear time computation of the abelian periods of a word. In J. Holub and J. Žďárek, editors, *Proceedings of the Prague Stringology Conference 2012*, pages 103–110, Czech Technical University in Prague, Czech Republic, 2012.
16. M. L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with O(1) worst case access time. *J. ACM*, 31(3):538–544, 1984.
17. T. Gagie, D. Hermelin, G. M. Landau, and O. Weimann. Binary jumbled pattern matching on trees and tree-like structures. *CoRR*, abs/1301.6127, 2013. Accepted to ESA 2013.
18. T. Kociumaka, J. Radoszewski, and W. Rytter. Fast algorithms for abelian periods in words and greatest common divisor queries. In N. Portier and T. Wilke, editors, *STACS*, volume 20 of *LIPIcs*, pages 245–256. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.
19. V. Lacroix, C. G. Fernandes, and M.-F. Sagot. Motif search in graphs: Application to metabolic networks. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 3(4):360–368, 2006.
20. T. M. Moosa and M. S. Rahman. Indexing permutations for binary strings. *Inf. Process. Lett.*, 110(18-19):795–798, 2010.
21. T. M. Moosa and M. S. Rahman. Sub-quadratic time and linear space data structures for permutation matching in binary strings. *J. Discrete Algorithms*, 10:5–9, 2012.