Composite Repetition-Aware Data Structures

Djamal Belazzougui[1,2(✉)], Fabio Cunial[1,2], Travis Gagie[1,2], Nicola Prezza[3], and Mathieu Raffinot[4]

[1] Department of Computer Science, University of Helsinki, Helsinki, Finland
djamal.belazzougui@cs.helsinki.fi
[2] Helsinki Institute for Information Technology, Helsinki, Finland
[3] Department of Mathematics and Computer Science, University of Udine, Udine, Italy
[4] LIAFA, Paris Diderot University, Paris 7, France

Abstract. In highly repetitive strings, like collections of genomes from the same species, distinct measures of repetition all grow sublinearly in the length of the text, and indexes targeted to such strings typically depend only on one of these measures. We describe two data structures whose size depends on multiple measures of repetition at once, and that provide competitive tradeoffs between the time for counting and reporting all the exact occurrences of a pattern, and the space taken by the structure. The key component of our constructions is the run-length encoded BWT (RLBWT), which takes space proportional to the number of BWT runs: rather than augmenting RLBWT with suffix array samples, we combine it with data structures from LZ77 indexes, which take space proportional to the number of LZ77 factors, and with the compact directed acyclic word graph (CDAWG), which takes space proportional to the number of extensions of maximal repeats. The combination of CDAWG and RLBWT enables also a new representation of the suffix tree, whose size depends again on the number of extensions of maximal repeats, and that is powerful enough to support matching statistics and constant-space traversal.

1 Introduction

The space taken by compressed data structures for highly-repetitive strings is typically a function of a specific measure of repetition, for example the number z of factors in a Lempel-Ziv parsing [1,11], or the number r of runs in a Burrows-Wheeler transform [14]. For many such compressed data structures, computing all the occurrences of a pattern in the indexed string is a bottleneck. In this paper we explore the advantages of *combining data structures that depend on distinct measures of repetition*. Specifically, we describe a data structure that takes approximately $O(z+r)$ words of space, and that reports all the occurrences

Travis Gagie—Supported by the Academy of Finland.
This work was partially supported by Academy of Finland under grant 284598 (Center of Excellence in Cancer Genetics Research).

F. Cicalese et al. (Eds.): CPM 2015, LNCS 9133, pp. 26–39, 2015.
DOI: 10.1007/978-3-319-19929-0_3

of a pattern of length m in $O(m(\log\log n + \log z) + \texttt{pocc}\log^{\epsilon} z + \texttt{socc}\log\log n)$ time, where n is the length of the string and pocc and socc are the number of primary and of secondary occurrences, respectively (see Sect. 2.2 for definitions). This compares favorably to the $O(m^2h + (m + \texttt{occ})\log z)$ reporting time of LZ77 indexes [11], where h is the height of the parse tree. It also compares favorably in space to solutions based on run-length encoded BWT (RLBWT) and suffix array samples [14], which take $O(n/k + r)$ words of space to achieve $O(m\log\log n + k \cdot \texttt{occ}\log\log n)$ reporting time, where k is a sampling rate.

We also introduce a new measure of the repetitiveness of a string, the number e of right extensions of maximal repeats, which is related to the number of arcs in the compact directed acyclic word-graph (CDAWG) and which is an upper bound on r and z. We show a data structure whose size depends on e and that reports all the occ occurrences of a pattern of length m in a string of length n in $O(m\log\log n + \texttt{occ})$ time. The main component of our constructions is the RLBWT, which we use to count the number of occurrences of a pattern, and which we combine with the CDAWG and with data structures from LZ indexes, rather than with suffix array samples, for reporting. Similar combinations have already appeared in the literature, but their space has been related to statistical compressibility rather than to the number of repetitions: for example, an FM-index has already been combined with an LZ78 self-index to achieve faster search or reporting [1,7], but the size of the resulting data structure depends on k-th order empirical entropy. Bounds in terms of k-th order empirical entropy have redundancy terms that depend exponentially on k, so they cannot capture compressibility based on long repetitions.

Combining the RLBWT with the CDAWG enables also a new representation of the suffix tree, which takes space proportional to $e+e^{\ell}$ (where e^{ℓ} is the number of left extensions of maximal repeats) and which supports a number of operations in $O(\log\log n)$ time. Among other properties, this new representation allows computing the matching statistics of a pattern of length m in $O(m\log\log n)$ time. Our constructions are targeted to highly-repetitive strings, like large databases of similar genomes, in which all the measures of repetition on which our data structures depend grow sublinearly in the size of the database (see Fig. 1 for an example). In a future paper we will provide a full experimental comparison of our results against other data structures for pattern matching in highly-repetitive strings.

2 Preliminaries

Let $\Sigma = [1..\sigma]$ be an integer alphabet, let $\# = 0 \notin \Sigma$ be a separator, and let $T = [1..\sigma]^{n-1}\#$ be a string. We denote the reverse of T by $\overline{T}$. Given a substring W of T, let $\mathcal{P}_T(W)$ be the set of all starting positions of W in the circular version of T. A *repeat* W is a string that satisfies $|\mathcal{P}_T(W)| > 1$. We denote by $\Sigma^{\ell}_T(W)$ the set of characters $\{a \in [0..\sigma] : |\mathcal{P}_T(aW)| > 0\}$ and by $\Sigma^{r}_T(W)$ the set of characters $\{b \in [0..\sigma] : |\mathcal{P}_T(Wb)| > 0\}$. A repeat W is *right-maximal* (respectively, *left-maximal*) iff $|\Sigma^{\ell}_T(W)| > 1$ (respectively, iff $|\Sigma^{r}_T(W)| > 1$).

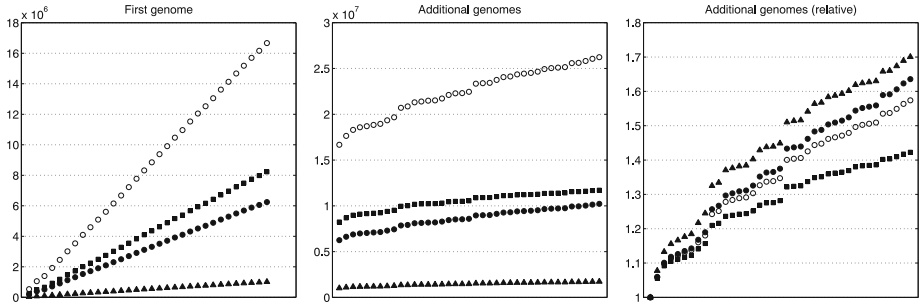

Fig. 1. Growth of the number of maximal repeats $|\mathcal{M}_T|$ (black circles), of $|\mathcal{E}^r_T \cup \mathcal{F}^r_T|$ (white circles, e in the introduction), of the number of runs in BWT $|\mathcal{R}_T|$ (squares, r in the introduction), and of $|\mathcal{Z}_T|$ (triangles, z in the introduction) in a concatenation T of 39 highly similar *Saccharomyces cerevisiae* genomes [8] (see Sect. 2 for definitions). Left: growth inside the first genome of the database. Center: growth after the addition of each genome (one sample per genome). Right: the same as the plot in the center, but with each curve normalized by its first sample. $|\mathcal{E}^\ell_T \cup \mathcal{F}^\ell_T|$, $|\mathcal{R}_{\overline{T}}|$ and $|\mathcal{Z}_{\overline{T}}|$ are not shown since they behave approximately as their symmetrical counterparts.

It is well known that T can have at most $n-1$ right-maximal substrings and at most $n-1$ left-maximal substrings. A *maximal repeat* of T is a repeat that is both left- and right-maximal: we call $\mathcal{M}_T$ the set of all maximal repeats of T. A maximal repeat W can be seen as a set of right-maximal substrings of T, and specifically as the set of all right-maximal strings $W[i..|W|]$ for $i \in [1..k]$ that are not left-maximal, and such that $W[k+1..|W|]$ is left-maximal.

For reasons of space we assume the reader to be familiar with the notion of *suffix tree* $\mathsf{ST}_T = (V, E)$ of T, which we do not define here. We denote by $\ell(\gamma)$, or equivalently by $\ell(u, v)$, the label of edge $\gamma = (u, v) \in E$, and we denote by $\ell(v)$ the string label of node $v \in V$. It is well known that a substring W of T is right-maximal (respectively, left-maximal) iff $W = \ell(v)$ for some internal node v of ST_T (respectively, iff $W = \overline{\ell(v)}$ for some internal node v of $\mathsf{ST}_{\overline{T}}$). We assume the reader to be familiar with the notion of *suffix link* connecting a node v with $\ell(v) = aW$ for some $a \in [0..\sigma]$ to a node w with $\ell(w) = W$: we say that $w = \texttt{suffixLink}(v)$ in this case. Here we just recall that inverting the direction of all suffix links yields the so-called *explicit Weiner links*. Given an internal node v and a symbol $a \in [0..\sigma]$, it might happen that string $a\ell(v)$ does occur in T, but that it is not right-maximal, i.e. it is not the label of any internal node: all such left extensions of internal nodes that end in the middle of an edge are called *implicit Weiner links*. An internal node can have more than one outgoing Weiner link, and all such Weiner links have distinct labels.

The *compact directed acyclic word graph* of a string T (denoted by CDAWG_T in what follows) is the minimal compact automaton representing the set of suffixes of a given string [3,6]. It can be seen as the minimization of ST_T, in which all leaves are merged to the same node (the sink) that represents T itself, and in which all nodes except the sink are in one-to-one correspondence with the

maximal repeats of T [16]. Since a maximal repeat corresponds to a set of right-maximal substrings, CDAWG_T can be built by putting in the same equivalence class all nodes of ST_T that belong to the same maximal unary path of explicit Weiner links.

For reasons of space we assume the reader to be familiar with the notion and uses of the Burrows-Wheeler transform of T, including the C array and backward searching. In this paper we use BWT_T to denote the BWT of T, and we use $\mathtt{range}(W) = [\mathtt{sp}(W)..\mathtt{ep}(W)]$ to denote the lexicographic interval of a string W in a BWT that is implicit from the context. We say that $\mathsf{BWT}_T[i..j]$ is a *run* iff $\mathsf{BWT}_T[k] = c \in [0..\sigma]$ for all $k \in [i..j]$, and moreover if any substring $\mathsf{BWT}_T[i'..j']$ such that $i' \leq i$, $j' \geq j$, and either $i' \neq i$ or $j' \neq j$, contains at least two distinct characters. It is well known that repetitions in T tend to be converted into runs of BWT_T. We denote by $\mathcal{R}_T$ the set of all triplets (c, i, j) such that $\mathsf{BWT}_T[i..j]$ is a run of character c, and we use r_T and $\overline{r}_T$ as shorthands for $|\mathcal{R}_T|$ and $|\mathcal{R}_{\overline{T}}|$, respectively.

The *LZ77 factorization* of T [20] is the greedy decomposition $T_1T_2\cdots T_z$ of T obtained as follows. Assume that T is virtually preceded by the σ distinct characters in its alphabet, and assume that $T_1T_2\cdots T_i$ has already been computed for some prefix of length k of T: then, T_{i+1} is the longest prefix of $T[k+1..n]$ such that there is a $j \leq k$ that satisfies $T[j..j+|T_{i+1}|-1] = T_{i+1}$. We denote by $\mathcal{Z}_T$ the set of pairs (T_i, p_i) for all $i \in [1..z]$, where p_i is the starting position of T_i in T, and we use z_T as a shorthand for $|\mathcal{Z}_T|$. From now on, we drop subscripts whenever the string T they specify is clear from the context.

2.1 Relationships Among Maximal Repeats, Runs in BWT, and LZ Factors

Clearly $|\mathcal{R}|$ can be as small as two, e.g. in string $\mathtt{0}^{n-1}\#$, and as large as $\Theta(n)$, e.g. in the string of length n that contains exactly n distinct characters, or in a de Bruijn string of order $k > 1$ on a binary alphabet: this string of length $\sigma^k + k - 1$ contains all the distinct k-mers, thus the interval of every $(k-1)$-mer in BWT_T contains exactly σ distinct characters, and the number of runs in BWT_T is thus at least $\sigma^{k-1}(k-1)$. It is known that $|\mathcal{Z}|$ is $O(n/\log_\sigma n)$ [12], and it can be constant, e.g. in $\mathtt{0}^{n-1}\#$. Conversely, $|\mathcal{M}|$ can be zero, e.g. in a string of length n that contains exactly n distinct characters, and it can be $\Theta(n)$ in the worst case, e.g. in string $\mathtt{0}^{n-1}\#$. When maximal repeats exist, the number of *right extensions of maximal repeats* $\sum_{W\in\mathcal{M}} |\Sigma^r(W)|$ is $\Omega(\log n)$, and this lower bound is matched by Fibonacci strings and by Thue-Morse strings of length n, whose CDAWG contains $O(\log n)$ nodes [15,17]. Both $|\mathcal{M}|/|\mathcal{R}|$ and $|\mathcal{M}|/|\mathcal{Z}|$ can be $\Theta(n)$, for example in the already mentioned $\mathtt{0}^{n-1}\#$. $|\mathcal{R}|/|\mathcal{Z}|$ can be $\Theta(\log n)$, e.g. in the already mentioned de Bruijn string T of order k, which has $\Theta(n/\log_\sigma n)$ LZ factors. However, $|\mathcal{M}|$, $|\mathcal{R}|$ and $|\mathcal{Z}|$ can all grow at the same asymptotic rate in the same family of strings. Consider e.g. string $T = \mathtt{0}^1\mathtt{1}\mathtt{0}^2\mathtt{1}\cdots\mathtt{0}^x\mathtt{1}\#$ of length $x(x+3)/2+1$. Clearly $|\mathcal{Z}| = x+3$, and $|\mathcal{M}| = 3(x-1)$ since the maximal repeats of T are only the substrings $\mathtt{0}^i\mathtt{1}$ for $i \in [1..x-1]$, $\mathtt{0}^j$ for $j \in [1..x-1]$,

and $\mathtt{0}^{k-1}\mathtt{10}^k$ for $k \in [2..x-1]$. Replacing # with a new block $\mathtt{0}^{x+1}\mathtt{1}\#$ in string T creates two new runs for every $x > 1$, thus $|\mathcal{R}| = 2x$ for $x > 1$.

Recall that a substring W of T is a maximal repeat iff $W = \ell(v)$ for some internal node v of $\mathsf{ST}_T = (V, E)$, and moreover if there are at least two Weiner links from v. Since the set of all left-maximal substrings of T is closed under the prefix operation, there is a bijection between $\mathcal{M}$ and the nodes that lie on the paths of ST_T that start from the root and that end at nodes labeled by maximal repeats defined as follows:

Definition 1. *A maximal repeat W of a string $T \in [1..\sigma]^{n-1}\#$ is* rightmost *if no string WV with $V \in [0..\sigma]^+$ is left-maximal in T.*

We denote the set of rightmost maximal repeats of T by $\mathcal{M}^r_T$. We also denote by $\mathcal{E}^r_T$ the set of edges of ST_T that connect pairs of nodes labeled by maximal repeats, and we denote by $\mathcal{F}^r_T$ the set of edges (v, w) in ST_T such that $\ell(v) \in \mathcal{M}_T$ and $\ell(w) \notin \mathcal{M}_T$. We use $\mathcal{M}^\ell_T$, $\mathcal{E}^\ell_T$ and $\mathcal{F}^\ell_T$ to denote symmetrical concepts in $\mathsf{ST}_{\overline{T}}$, and we use e_T and e^ℓ_T as shorthands for $|\mathcal{E}^r_T| + |\mathcal{F}^r_T|$ and for $|\mathcal{E}^\ell_T| + |\mathcal{F}^\ell_T|$, respectively. Clearly $\mathcal{E}^r$ and $\mathcal{F}^r$ are the image of explicit and implicit Weiner links of $\mathsf{ST}_{\overline{T}}$:

Lemma 1. *Let $\mathsf{ST}_T = (V, E)$. There is a bijection between $\mathcal{E}^r_T$ and the set of all explicit Weiner links from nodes of $\mathsf{ST}_{\overline{T}}$ that correspond to maximal repeats of T. There is a bijection between $\mathcal{F}^r_T$ and the set of all implicit Weiner links from nodes of $\mathsf{ST}_{\overline{T}}$ that correspond to maximal repeats of T.*

The proof of Lemma 1 is provided in the appendix. It is clear that the set of suffix tree edges $\mathcal{E}^r_T \cup \mathcal{F}^r_T$ is in one-to-one correspondence with the set of all arcs of CDAWG_T. This set of edges is also related to runs in BWT_T:

Theorem 1. $|[0..\sigma] \setminus \cup_{W \in \mathcal{M}^r_T} \Sigma^\ell_T(W)| + \sum_{W \in \mathcal{M}^r_T} |\Sigma^\ell_T(W)| - |\mathcal{M}^r_T| + 1 \leq |\mathcal{R}_T| \leq |\mathcal{F}^r_T|$.

Proof. The root of ST_T is a maximal repeat, thus the destinations of all edges in $\mathcal{F}^r$ partition all leaves of ST_T into disjoint subtrees, or equivalently they partition the entire BWT_T in disjoint blocks. Since every such block is the interval in BWT_T of some string that is not left-maximal, all characters of BWT_T in the same block are identical, thus the number of runs in BWT_T cannot be bigger than $|\mathcal{F}^r|$.

The interval of a string $W \in \mathcal{M}^r$ in BWT_T contains exactly $|\Sigma^\ell(W)|$ distinct characters, and at most one of them is identical to the character that precedes the largest suffix of T smaller than W in lexicographic order (note that such suffix might not be prefixed by any string in $\mathcal{M}^r$). Thus, the number of runs in BWT_T is at least $\sum_{W \in \mathcal{M}^r} |\Sigma^\ell(W)| - |\mathcal{M}^r| + 1$. Factor $[0..\sigma] \setminus \cup_{W \in \mathcal{M}^r} \Sigma^\ell_T(W)$ in the claim takes into account symbols of T that never occur to the left of strings in $\mathcal{M}^r$. □

A symmetrical argument holds for $\mathcal{R}_{\overline{T}}$. The set of arcs in CDAWG_T is also related to the LZ factorization of T:

Theorem 2. $|\mathcal{Z}_T| \leq |\mathcal{E}^r_T \cup \mathcal{F}^r_T|$

Proof. Let $T = T_1 T_2 \dots T_z$ be the LZ factorization of T, and let $p_1, p_2, \dots, p_z$ be the sequence such that p_i is the starting position of factor T_i in T. Every factor is a right-maximal substring of T, but it is not necessarily left-maximal: let W_i be a suffix of $T[1..p_i - 1]$ such that $W_i T_i$ is both right-maximal and left-maximal, and assume that we assign T_i to the edge (v, w) in $\mathcal{E}^r_T \cup \mathcal{F}^r_T$ such that $\ell(v) = W_i T_i$, $v = \texttt{parent}(w)$, and the first character of T_{i+1} equals the first character of $\ell(v, w)$. Assume that there is some $j > i$ for which we assign T_j to the same maximal repeat $W_i T_i$. Then, the first character of T_{j+1} must be different from the first character of T_{i+1}, otherwise factor T_j would have been longer. It follows that every LZ factor can be assigned to a distinct element of $\mathcal{E}^r_T \cup \mathcal{F}^r_T$. □

The gap between r and e, and between z and e, is apparent from Fig. 1 (center). However, all these measures seem to grow at the same relative rate in practice (right panel).

2.2 Repetition-Aware Data Structures

Given a string $T \in [1..\sigma]^{n-1}\#$, we call *run-length encoded BWT* any representation of BWT_T that takes $O(|\mathcal{R}_T|)$ words of space, and that supports rank and select operations: see for example [13,14,18]. Let $\mathcal{R}_T$ be a set of triplets (c, i, j) such that $\mathsf{BWT}_T[i..j]$ is a run of character c. It is easy to implement rank in $O(\log \log n)$ time, by encoding $\mathcal{R}_T$ as $\sigma + 1$ predecessor data structures [19], each of which stores the second component of all triplets with the same first component. For every such second component i, we also store in an array the sum of all occurrences of c up to i, exclusive. To implement select in $O(\log \log n)$ time, we can similarly encode $\mathcal{R}_T$ as $\sigma + 1$ predecessor data structures, each of which stores value $\texttt{rank}_c(\mathsf{BWT}_T, i-1)$ for all triplets (c, i, j) with the same value of c. We also store the value of i for every such triplet. We denote the run-length encoded BWT of T by RLBWT_T.

For reasons of space we assume the reader to be familiar with LZ77-indexes: see e.g. [9,10]. Here we just recall that a *primary occurrence* of a pattern P in a string $T \in [1..\sigma]^{n-1}\#$ is one that crosses a phrase boundary in the LZ77 factorization $T_1 T_2 \cdots T_z$ of T. All other occurrences are called *secondary*. Once we have determined all primary occurrences, locating secondary occurrences reduces to two-sided range reporting and takes $O(\mathsf{occ} \log \log n)$ time with a data structure that takes $O(z)$ words of space [10]. To locate primary occurrences, we can use a data structure for four-sided range reporting on a $z \times z$ grid, with a marker at (x, y) if the xth LZ factor in lexicographic order is preceded in the text by the lexicographically yth reversed prefix ending at a phrase boundary. This data structure takes $O(z)$ words of space, and it returns all the phrase boundaries immediately followed by a factor in the specified range, and immediately preceded by a reversed prefix in the specified range, in $O((1 + k) \log^{\epsilon} z)$ time, where k is the number of phrase boundaries reported [4].

3 Combining Runs in BWT and LZ Factors

In this section we describe how to combine data structures whose size depends on the number of LZ factors of a string $T \in [1..\sigma]^{n-1}\#$, and data structures whose size depends on the number of runs in BWT_T, to report all the occurrences of a pattern in T. To do so, we first need to solve the following subproblem. Let $\mathsf{ST}_T = (V, E)$ be the suffix tree of T, and let $V' = \{v_1, v_2, \ldots, v_k\} \subseteq V$ be a subset of the nodes of ST_T. Consider the list of node labels $L = \ell(v_1), \ell(v_2), \ldots, \ell(v_k)$, sorted in lexicographic order. Given a string $W \in [0..\sigma]^*$, we want to implement function $\mathbb{I}(W, V')$ that returns the (possibly empty) interval of W in L. The following lemma describes how to do this in $O(k)$ words of space:

Lemma 2. *Let $T \in [1..\sigma]^{n-1}\#$ be a string, and let V' be a subset of k nodes of its suffix tree, represented as intervals in BWT_T. Given the interval $[i..j]$ of a string $W \in [0..\sigma]^*$ in BWT_T, there is a data structure that takes $O(k)$ words of space and that computes $\mathbb{I}(W, V')$ in $O(\log k)$ time.*

Proof. Let $F[1..n]$ be a bitvector such that $F[i] = 1$ iff there is a node $v' \in V'$ such that $\texttt{range}(v') = [i..j]$. Similarly, let $L[1..n]$ be a bitvector such that $L[j] = 1$ iff there is a node $v' \in V'$ such that $\texttt{range}(v') = [i..j]$. Let α and β be the number of ones in F and L, respectively. We store in array $\texttt{first}[1..\alpha]$ (respectively, $\texttt{last}[1..\beta]$) the sorted positions of the ones in F (respectively, in L), using $O(k)$ words of space. Let $F'[1..\alpha]$ be the array such that $F'[i]$ equals the number of intervals $[p..q]$ such that p is the ith one in F and $[p..q] = \texttt{range}(v')$ for a node $v' \in V'$. Similarly, let $L'[1..\beta]$ be the array such that $L'[i]$ equals the number of intervals $[p..q]$ such that q is the ith one in L and $[p..q] = \texttt{range}(v')$ for a node $v' \in V'$. We represent F' and L' as prefix-sum arrays $\texttt{first}'[1..\alpha]$ and $\texttt{last}'[1..\beta]$ using $O(k)$ words of space, i.e. $\texttt{first}'[i] = \sum_{h=1}^{i} F'[h]$ and $\texttt{last}'[i] = \sum_{h=1}^{i} L'[h]$.

Let $\mathbb{I}(W, V') = [x..y]$. Given the interval $[i..j]$ of a string W in BWT_T, we find the corresponding interval $[i'..j']$ in array $\texttt{first}$ in $O(\log \alpha)$ time, using binary search on $\texttt{first}'$. Specifically, $i' = \min\{h \in [1..\alpha] : \texttt{first}'[h] \geq i\}$ and $j' = \max\{h \in [1..\alpha] : \texttt{first}'[h] \leq j\}$. If $j' < i'$ then W is not the prefix of a label of a node in V'. Otherwise, since all nodes $v' \in V'$ whose BWT interval starts inside $[i+1..j]$ are right extensions of W, we set $y = \sum_{h=1}^{j'} F'[h] = \texttt{first}'[j']$ in constant time. If $\texttt{first}[i'] \neq i$, i.e. if no interval of a node $v' \in V'$ starts at position i in BWT_T, then we can just set $x = 1 + \sum_{h=1}^{i'-1} F'[h] = 1 + \texttt{first}'[i'-1]$ in constant time and stop.

Otherwise, it could happen that just a (possibly empty) subset of all the nodes in V' whose interval starts at position i in BWT_T correspond to W or to right extensions of W: the intervals of such nodes necessarily end inside $[i..j]$. All the other intervals that start at position i could correspond instead to *prefixes* of W, and they necessarily end after position j in BWT_T. Thus, let $[i''..j'']$ be the interval in $\texttt{last}$ that corresponds to $[i..j]$: specifically, let $i'' = \min\{h \in [1..\beta] : \texttt{last}[h] \geq i\}$ and $j'' = \max\{h \in [1..\beta] : \texttt{last}[h] \leq j\}$. To determine the number of intervals that start at position i in BWT_T and that correspond

to prefixes of W, it suffices to compute the difference δ between the number of starting positions and the number of ending positions inside interval $[i..j]$, as follows: $\delta = \sum_{h=1}^{j'} F'[h] - \sum_{h=1}^{i'-1} F'[h] - \sum_{h=1}^{j''} L'[h] + \sum_{h=1}^{i''-1} L'[h]$. Then, $x = \sum_{h=1}^{i'-1} F'[h] + \delta + 1$. All such sums can be computed in constant time using the prefix-sum representations of F' ad L'.

If the interval of some node in V' starts at i and ends after j in BWT_T, then no interval can end at j and start before i, so δ is nonnegative. □

Consider now a factorization of T such that all factors are right-maximal substrings of T, and let V' be the set of nodes of ST_T that correspond to the distinct factors. To locate all the occurrences of a pattern that cross or end at a boundary between two factors, we just need an implementation of function $\mathbb{I}(W, V')$ and a pair of RLBWTs:

Lemma 3. *Let $T \in [1..\sigma]^{n-1}\#$ be a string, and let $T = T_1 T_2 \cdots T_z$ be a factorization of T in which all factors are right-maximal substrings. There is a data structure that takes $O(z + r_T + \overline{r}_T)$ words of space and that reports all the* occ *occurrences of a pattern $P \in [0..\sigma]^m$ that cross or end at a boundary between two factors of T, in $O(m(\log\log n + \log z) + \mathtt{occ}\log^{\epsilon} z)$ time.*

Proof. Let $p_1, p_2, \ldots, p_z$ be the sequence such that p_i is the starting position of factor T_i in T. The same occurrence of P in T can cover up to m boundaries between two factors, thus we organize the computation as follows. We consider every possible way to place *the rightmost boundary between two factors* in P, i.e. every possible split of P into two parts $P[1..k-1]$ and $P[k..m]$ for $k \in [1..m]$, such that $P[k..m]$ is either a factor or a proper prefix of a factor. For every such k, we use four-sided range reporting queries to list all the occurrences of P in T that conform to this split, as described in Sect. 2.2. The four-sided range reporting data structure represents the mapping between the lexicographic rank of a factor W among all the distinct factors of T, and the lexicographic ranks of all the reversed prefixes $\overline{T[1..p_i - 1]}$ such that $T_i = W$, among all the reversed prefixes of T that end at the last position of a factor. As described in Sect. 2.2, this data structure takes $O(z)$ words of space.

We encode sequence $p_1, p_2, \ldots, p_z$ implicitly, as follows: we use a bitvector $\mathtt{last}[1..n]$ such that $\mathtt{last}[i] = 1$ iff $\mathsf{SA}_{\overline{T}}[i] = n - p_j + 2$ for some $j \in [1..z]$, i.e. iff $\mathsf{SA}_{\overline{T}}[i]$ is the last position of a factor. We represent such bitvector as a predecessor data structure with partial ranks, using $O(z)$ words of space [19]. Then, we build the data structure described in Lemma 2, where V' is the set of loci in ST_T of all factors of T. This data structure takes $O(z)$ words of space, and together with $\mathtt{last}$, RLBWT_T and $\mathsf{RLBWT}_{\overline{T}}$, it is the output of our construction.

Given a pattern $P \in [0..\sigma]^m$, we first perform a backward search in RLBWT_T to determine the number of occurrences of P in T: if this number is zero, we stop. During this backward search, we store in a table the interval $[i_k..j_k]$ of $P[k..m]$ in BWT_T for every $k \in [2..m]$. Then, we compute the interval $[i'_{k-1}..j'_{k-1}]$ of $\overline{P[1..k-1]}$ in $\mathsf{BWT}_{\overline{T}}$ for every $k \in [2..m]$, using backward search in $\mathsf{RLBWT}_{\overline{T}}$: if $\mathtt{rank}_1(\mathtt{last}, j'_{k-1}) - \mathtt{rank}_1(\mathtt{last}, i'_{k-1} - 1) = 0$, then $P[1..k-1]$ never ends at the

last position of a factor, and we can discard this value of k. Otherwise, we convert $[i'_{k-1}..j'_{k-1}]$ to the interval $[\texttt{rank}_1(\texttt{last}, i'_{k-1}) + 1..\texttt{rank}_1(\texttt{last}, j'_{k-1})]$ of all the reversed prefixes of T that end at the last position of a factor. Rank operations on `last` can be implemented in $O(\log\log n)$ time using predecessor queries. We get the lexicographic interval of $P[k..m]$ in the list of all the distinct factors of T using operation $\mathbb{I}(P[k..m], V')$, in $O(\log z)$ time. We use such intervals to query the four-sided range reporting data structure. □

The algorithm described in Lemma 3 can be engineered in a number of ways in practice. Here we just apply it to the LZ factorization of T to find all the primary occurrences of P in T, and we use the strategy described in Sect. 2.2 to compute secondary occurrences, obtaining the key result of this section:

Theorem 3. *Let $T \in [1..\sigma]^{n-1}\#$ be a string, and let $T = T_1T_2\dots T_z$ be its LZ factorization. There is a data structure that takes $O(z + r_T + \overline{r}_T)$ words of space and that reports all the* `pocc` *primary occurrences and all the* `socc` *secondary occurrences of a pattern $P \in [0..\sigma]^m$ in $O(m(\log\log n + \log z) + \texttt{pocc}\log^{\epsilon} z + \texttt{socc}\log\log n)$ time.*

4 Combining Runs in BWT and Maximal Repeats

An alternative way to compute all the occurrences of a pattern in a string T consists in combining RLBWT_T with CDAWG_T, using an amount of space proportional to the number of right extensions of the maximal repeats of T:

Theorem 4. *Let $T \in [1..\sigma]^{n-1}\#$ be a string. There is a data structure that takes $O(e_T)$ words of space (or alternatively, $O(e^{\ell}_T)$ words of space) and that reports all the* `occ` *occurrences of a pattern $P \in [0..\sigma]^m$ in $O(m\log\log n + \texttt{occ})$ time.*

Proof. We build RLBWT_T and CDAWG_T. For every node v in the CDAWG, we store $|\ell(v)|$ in a variable $v.\texttt{length}$. Recall that an arc (v, w) of the CDAWG means that maximal repeat $\ell(w)$ can be obtained by extending maximal repeat $\ell(v)$ to the right and to the left. Thus, for every arc $\gamma = (v, w)$ of CDAWG_T, we store the first character of $\ell(\gamma)$ in a variable $\gamma.\texttt{char}$, and we store the length of the right extension implied by γ in a variable $\gamma.\texttt{right}$. The length $\gamma.\texttt{left}$ of the left extension implied by γ can be computed by $w.\texttt{length} - v.\texttt{length} - \gamma.\texttt{right}$. Clearly arcs of CDAWG_T that correspond to edges of ST_T in set $\mathcal{E}^r_T$ induce no left extension. For every arc of CDAWG_T that connects a maximal repeat W to the sink, we store just $\gamma.\texttt{char}$ and the starting position $\gamma.\texttt{pos}$ of string $W \cdot \gamma.\texttt{char}$ in T. The total space used by the CDAWG is clearly $O(e)$ words, and by Theorem 1 the space used by RLBWT_T is $O(|\mathcal{F}^r_T|)$ words. An alternative construction could use $\mathsf{CDAWG}_{\overline{T}}$ and $\mathsf{RLBWT}_{\overline{T}}$.

We use the RLBWT to count the number of occurrences of P in T in $O(m\log\log n)$ time: if this number is greater than zero, we use the CDAWG to report all the `occ` occurrences of P in T in $O(\texttt{occ})$ time, using the technique

sketched in [5]. Specifically, since we know that P occurs in T, we perform a blind search for P in the CDAWG, as is typically done with Patricia trees. We keep a variable i, initialized to zero, that stores the length of the prefix of P that we have matched so far, and we keep a variable j, initialized to one, that stores the starting position of P inside the last maximal repeat encountered during the search. For every node v in the CDAWG, we choose the arc γ such that $\gamma.\texttt{char} = P[i+1]$ in constant time using hashing, we increment i by $\gamma.\texttt{right}$, and we increment j by $\gamma.\texttt{left}$. If the search leads to the sink by an arc γ, we report $\gamma.\texttt{pos} + j$ and we stop. If the search leads to a node v that is associated with the maximal repeat W, we determine all the occurrences of W in T by performing a depth-first traversal of all the nodes in the CDAWG that are reachable from v, updating variables i and j as described above, and reporting $\gamma.\texttt{pos} + j$ for every arc γ that leads to the sink. The total number of nodes and arcs reachable from v is clearly $O(\texttt{occ})$. □

The combination of CDAWG_T and RLBWT_T can also be used to implement a repetition-aware representation of ST_T. We will apply the following property to support operations on ST_T:

Property 1. A maximal repeat $W = [1..\sigma]^m$ of T is the equivalence class of all the right-maximal strings $\{W[1..m], \ldots, W[k..m]\}$ such that $W[k+1..m]$ is left-maximal, and $W[i..m]$ is not left-maximal for all $i \in [2..k]$. Equivalently, the node v' of CDAWG_T with $\ell(v') = W$ is the equivalence class of the nodes $\{v_1, \ldots, v_k\}$ of ST_T such that $\ell(v_i) = W[i..m]$ for all $i \in [1..k]$, and such that $v_k, v_{k-1}, \ldots, v_1$ is a maximal unary path of Weiner links.

Thus, the set of right-maximal strings that belong to the equivalence class of a maximal repeat can be represented by a single integer k, and a right-maximal string can be identified by the maximal repeat W it belongs to, and by the length of the corresponding suffix of W. In BWT_T, the right-maximal strings in the same equivalence class enjoy the following additional properties:

Property 2. Let $\{W[1..m], \ldots, W[k..m]\}$ be the right-maximal strings that belong to the equivalence class of maximal repeat $W \in [1..\sigma]^m$, and let $\texttt{range}(W[i..m]) = [p_i..q_i]$ for $i \in [1..k]$. Then:

1. $|q_i - p_i + 1| = |q_j - p_j + 1|$ for all i and j in $[1..k]$.
2. $\mathsf{BWT}_T[p_i..q_i] = W[i-1]^{q_i - p_i + 1}$ for $i \in [2..k]$. Conversely, $\mathsf{BWT}_T[p_1..q_1]$ contains at least two distinct characters.
3. $p_{i-1} = C[c] + \texttt{rank}_c(\mathsf{BWT}_T, p_i)$ and $q_{i-1} = p_{i-1} + q_i - p_i$ for $i \in [2..k]$, where $c = W[i-1] = \mathsf{BWT}_T[p_i]$.
4. $p_{i+1} = \texttt{select}_c(\mathsf{BWT}_T, p_i - C[c])$ and $q_{i+1} = p_{i+1} + q_i - p_i$ for $i \in [1..k-1]$, where $c = W[i]$ is the character that satisfies $C[c] < p_i \leq C[c+1]$. This can be computed in $O(\log \log n)$ time using a predecessor data structure that uses $O(\sigma)$ words of space [19].
5. Let $c \in [0..\sigma]$, and let $\texttt{range}(W[i..m]c) = [x_i..y_i]$ for $i \in [1..k]$. Then, $x_i = p_i + x_1 - p_1$ and $y_i = p_i + y_1 - p_1$.

Table 1. Time complexities of two representations of ST_T: with intervals in BWT_T (row 1) and without intervals in BWT_T (row 2).

	stringDepth locateLeaf	isAncestor	parent nextSibling	child firstChild	suffixLink	weinerLink	edgeChar	nLeaves
1	$O(1)$	$O(1)$	$O(\log\log n)$	$O(1)$	$O(\log\log n)$	$O(\log\log n)$	$O(\log\log n)$	$O(1)$
2	$O(1)$		$O(\log\log n)$	$O(1)$	$O(1)$			

The final property we will exploit relates the equivalence class of a maximal repeat to the equivalence classes of its in-neighbors in the CDAWG:

Property 3. Let w be a node in CDAWG_T with $\ell(w) = W \in [1..\sigma]^m$, and let $\mathcal{S}_w = \{W[1..m], \ldots, W[k..m]\}$ be the right-maximal strings that belong to the equivalence class of node w. Let $\{v^1, \ldots, v^t\}$ be the in-neighbors of w in CDAWG_T, and let $\{V^1, \ldots, V^t\}$ be their labels. Then, $\mathcal{S}_w$ is partitioned into t disjoint sets $\mathcal{S}_w^1, \ldots, \mathcal{S}_w^t$ such that $\mathcal{S}_w^i = \{W[x^i+1..m], W[x^i+2..m], \ldots, W[x^i + |\mathcal{S}_{v^i}|..m]\}$, and the right-maximal string $V^i[p..|V^i|]$ labels the parent of the locus of the right-maximal string $W[x^i + p..m]$ in ST_T.

Proof. It is clear that the parent in ST_T of every right-maximal string in the equivalence class of node w belongs to the equivalence class of an in-neighbor of w: we focus here just on showing that the in-neighbors of w induce a partition on the equivalence class of w. Assume that the character that labels arc $\gamma = (v^i, w)$ in the CDAWG is c. Since arc γ exists, we can factorize W as $X^iV^iY^i$, where $Y^i[1] = c$, and we know that no prefix of V^iY^i longer than V^i is right-maximal, and that no suffix of W longer than $|V^iY^i|$ is left-maximal. Consider any suffix $V^i[p..|V^i|]$ of V^i that belongs to the equivalence class of V^i: if $p > 1$, then $W[|X^i|+p..m]$ is not left-maximal, thus $W[|X^i|+p..m]$ belongs to the equivalence class of W. Its prefix $V^i[p..|V^i|]$ is right-maximal, and no longer prefix is right-maximal. Indeed, assume that string $V^i[p..|V^i|]Z^i$ is right-maximal for some prefix Z^i of Y^i. Since $V^i[p..|V^i|]$ is not left-maximal, then string $V^i[p..|V^i|]Z^i$ is not left-maximal either, and this implies that V^iZ^i is right-maximal, contradicting the hypothesis. Thus, string $V^i[p..|V^i|]$ labels the parent of the locus of string $W[|X^i| + p..m]$ in ST_T. If $p = 1$ and V^iY^i is not left-maximal, the same argument applies. If V^iY^i is left-maximal, then $W = V^iY^i$, and since no right-maximal prefix of W longer than V^i exists, we have that V^i labels the parent of the locus of W in ST_T. □

Combining Properties 1, 2 and 3, we obtain the following results:

Theorem 5. *Let $T \in [1..\sigma]^{n-1}\#$ be a string. There are two implementations of ST_T that take $O(e_T + e_T^\ell)$ words of space each, and that support the operations in Table 1 with the specified time complexities.*

Proof. We build RLBWT_T and CDAWG_T, and we annotate the latter as described in Theorem 4, with the only difference that arcs that connect a maximal repeat to the sink are annotated with character and length like all other arcs. We store

in every node v of the CDAWG the number $v.\texttt{size}$ of right-maximal strings that belong to its equivalence class, the interval $[v.\texttt{first}..v.\texttt{last}]$ of $\ell(v)$ in BWT_T, a linear-space predecessor data structure [19] on the boundaries induced on the equivalence class of v by its in-neighbors (see Observation 3), and pointers to the in-neighbor that corresponds to the interval associated with each boundary. Finally, we add to the CDAWG all suffix links (v, w) from ST_T such that both v and w are maximal repeats, and the corresponding explicit Weiner links.

We represent a node v of ST_T as a tuple $\texttt{id}(v) = (v', |\ell(v)|, i, j)$, where v' is the node in CDAWG_T that corresponds to the equivalence class of v, and $[i..j]$ is the interval of $\ell(v)$ in BWT_T. Thus, operation `stringDepth` can be implemented in constant time, and if v is a leaf, the second component of $\texttt{id}(v)$ is its starting position in T. Operation `isAncestor` can be implemented by testing the containment of the corresponding intervals in BWT_T. To implement operation `suffixLink`, we first check whether $|\ell(v)| = v'.\texttt{length} - v'.\texttt{size} + 1$: if so, we take the suffix link (v', w') from v' and we return $(w', w'.\texttt{length}, w'.\texttt{first}, w'.\texttt{last})$. Otherwise, we return $(v', |\ell(v)| - 1, i', j')$, where $[i'..j']$ is computed as described in point 2 of Property 2. To implement `weinerLink` for some character c, we first check whether $|\ell(v)| = v'.\texttt{length}$: if so, we take the Weiner link (v', w') from v' labeled by character c (if any), and we return $(w', w'.\texttt{length} - w'.\texttt{size} + 1, i', j')$, where $[i'..j']$ is computed by taking a backward step with character c from $[v'.\texttt{first}..v'.\texttt{last}]$. Otherwise, we check whether $\mathsf{BWT}_T[i] = c$: if so, we return $(v', |\ell(v)| + 1, i', j')$, where $[i'..j']$ is computed as described in point 2 of Property 2.

To implement `child` for some character c, we follow the arc $\gamma = (v', w')$ in the CDAWG labeled by c (see Observation 3), and we return tuple $(w', |\ell(v)| + \gamma.\texttt{right}, i', j')$, where $[i'..j']$ is computed as described in point 2 of Property 2. To implement `parent` we exploit Property 2, i.e. we determine the partition of the equivalence class of v' that contains v by searching the predecessor of value $|\ell(v)|$ in the set of boundaries of v': this can be done in $O(\log\log n)$ time [19]. Let $\gamma = (u', v')$ be the arc that connects to v' the in-neighbor u' associated with the partition that contains v: we return tuple $(u', |\ell(v)| - \gamma.\texttt{right}, i', j')$, where $i' = i - v'.\texttt{first} + u'.\texttt{first}$ and $j' = j + u'.\texttt{last} - v'.\texttt{last}$ as described in point 2 of Property 2. Operation `nextSibling` can be implemented in the same way.

We read the label of an edge γ of ST_T in $O(\log\log n)$ time per character (operation `edgeChar`), by storing $\mathsf{RLBWT}_{\overline{T}}$ and the interval in $\mathsf{BWT}_{\overline{T}}$ of the reverse of the maximal repeat that corresponds to every node of the CDAWG. By removing from $\texttt{id}(v)$ the interval of $\ell(v)$ in BWT_T, we can implement `stringDepth`, `child`, `firstChild` and `suffixLink` in constant time, and `parent` and `nextSibling` in $O(\log\log n)$ time. □

Corollary 1. *Let $T \in [1..\sigma]^{n-1}\#$ be a string. There is an implementation of ST_T that takes $O(e_T + e_T^{\ell})$ words of space, that computes the matching statistics of a pattern $S \in [1..\sigma]^m$ with respect to T in $O(m \log\log n)$ time, and that can be traversed in $O(n \log\log n)$ time and in a constant number of words of space.*

Proof. We combine the implementation in the first row of Table 1 with the folklore algorithm for matching statistics, that issues `suffixLink` and `child` operations

on ST_T, and that reads the label of some edges of ST_T. For traversal, we combine the implementation in the second row of Table 1 with the folklore algorithm that issues just `firstChild`, `parent` and `nextSibling` operations. □

By storing $\mathsf{RLBWT}_{\overline{T}}$ in addition to RLBWT_T, and by adding to $\mathtt{id}(v)$ the interval of $\overline{\ell(v)}$ in $\mathsf{BWT}_{\overline{T}}$, we can also implement a bidirectional index on T like those described in [2], that supports the left and right extension of a string with any character in $O(\log \log n)$ time and that takes $O(e + e^{\ell})$ words of space.

References

1. Arroyuelo, D., Navarro, G., Sadakane, K.: Stronger Lempel-Ziv based compressed text indexing. Algorithmica **62**(1–2), 54–101 (2012)
2. Belazzougui, D., Cunial, F., Kärkkäinen, J., Mäkinen, V.: Versatile succinct representations of the bidirectional burrows-wheeler transform. In: Bodlaender, H.L., Italiano, G.F. (eds.) ESA 2013. LNCS, vol. 8125, pp. 133–144. Springer, Heidelberg (2013)
3. Blumer, A., Blumer, J., Haussler, D., McConnell, R., Ehrenfeucht, A.: Complete inverted files for efficient text retrieval and analysis. J. ACM **34**(3), 578–595 (1987)
4. Chan, T.M., Larsen, K.G., Pătraşcu, M.: Orthogonal range searching on the RAM, revisited. In: Proceedings of SoCG, pp. 1–10 (2011)
5. Crochemore, M., Hancart, C.: Automata for matching patterns. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 2, pp. 399–462. Springer, Heidelberg (1997)
6. Crochemore, M., Vérin, R.: Direct construction of compact directed acyclic word graphs. In: Apostolico, A., Hein, J. (eds.) Proceedings of CPM. LNCS, vol. 1264, pp. 116–129. Springer, Heidelberg (1997)
7. Ferragina, P., Manzini, G.: Indexing compressed text. J. ACM **52**(4), 552–581 (2005)
8. P. Ferragina and G. Navarro. Pizza&Chili repetitive corpus. http://pizzachili.dcc.uchile.cl/repcorpus.html. Accessed on 25 January 2015
9. Gagie, T., Gawrychowski, P., Kärkkäinen, J., Nekrich, Y., Puglisi, S.J.: LZ77-based self-indexing with faster pattern matching. In: Pardo, A., Viola, A. (eds.) LATIN 2014. LNCS, vol. 8392, pp. 731–742. Springer, Heidelberg (2014)
10. Kärkkäinen, J., Ukkonen, E.: Lempel-Ziv parsing and sublinear-size index structures for string matching. In: Proceedings of WSP, pp. 141–155 (1996)
11. Kreft, S., Navarro, G.: On compressing and indexing repetitive sequences. Theor. Comput. Sci. **483**, 115–133 (2013)
12. Lempel, A., Ziv, J.: On the complexity of finite sequences. IEEE Trans. Info. Theory **22**(1), 75–81 (1976)
13. Mäkinen, V., Navarro, G.: Succinct suffix arrays based on run-length encoding. In: Apostolico, A., Crochemore, M., Park, K. (eds.) CPM 2005. LNCS, vol. 3537, pp. 45–56. Springer, Heidelberg (2005)
14. Mäkinen, V., Navarro, G., Sirén, J., Välimäki, N.: Storage and retrieval of highly repetitive sequence collections. J. Comput. Biol. **17**(3), 281–308 (2010)
15. Radoszewski, J., Rytter, W.: On the structure of compacted subword graphs of ThueMorse words and their applications. J. Discret. Algorithms **11**, 15–24 (2012)
16. Raffinot, M.: On maximal repeats in strings. Inform. Process. Lett. **80**(3), 165–169 (2001)

17. Rytter, W.: The structure of subword graphs and suffix trees of Fibonacci words. Theoret. Comput. Sci. **363**(2), 211–223 (2006)
18. Sirén, J., Välimäki, N., Mäkinen, V., Navarro, G.: Run-length compressed indexes are superior for highly repetitive sequence collections. In: Amir, A., Turpin, A., Moffat, A. (eds.) SPIRE 2008. LNCS, vol. 5280, pp. 164–175. Springer, Heidelberg (2008)
19. Willard, D.E.: Log-logarithmic worst-case range queries are possible in space $\Theta(N)$. Inform. Process. Lett. **17**(2), 81–84 (1983)
20. Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. IEEE Trans. Info. Theory **23**(3), 337–343 (1977)