Locating All Tandem Repeat Families in a Sequence*

Donald Adjeroh and Jianan Feng

Abstract. **We present a new data structure called the BSCP (block sorted common prefix), and its tree representation, called the BSCP tree. We also introduce the notion of *PTR family* – a biologically motivated description and representation of the tandem repetitions in a sequence. The PTR family implicitly encodes each distinct primitive tandem repeat in the sequence as its part. Based on the BSCP tree, we describe a method to locate all the primitive tandem repeat families in an input sequence *T*. The proposed method requires average space and time complexity in $O(u)$, where $u = |T|$.**

1. Introduction

Repetition structures represent an important characteristic of genomic sequences. Although the precise biological function of these repetition structures is still a topic of intensive debate, it is, however, well known that the redundancy due to the repetition structures provides some form of stability for the genome. Tandem repeats in particular play a major role in various regulation mechanisms in the genome, such as in protein binding [Richards93hys]. They have also been linked with different recombination hot spots [Majewski2000o]. Repetition structures have been implicated in various diseases and genetic disorders. For instance, the triplet repeats $(CTG)_n/(CAG)_n$ have been associated with the Hutington's disease, while the hairpins formed in $(CGG)_n/(CCG)_n$ repeats have been linked to the Fragile-X mental retardation syndrome [Bat97ka]. Sinden et al [Sniden2002popls] identified fourteen of such genetic diseases that are linked with triplet repeats. An important observation for computational analysis of such repetition structures is that, in every single case listed, the susceptibility to (or incidence of) the disease critically depends on the ***number*** of copies (i.e. the copy exponents in the repeat), and *how many* times the triple repeat occurs with a given exponent.

In this work, we introduce a novel data structure called the ***block sorted common prefix***, BSCP. The BSCP provides a compact and complete representation of the information in the SCP data structure described in [Adjeroh2003f]. Based on the BSCP, we can address various important questions about tandem repeats in a simple manner. For instance, the number of distinct tandem repeats in a sequence (i.e. the size of the tandem repeat vocabulary) [Stoye2002g]; the number of different copy exponents for each distinct primitive tandem repeat [Kolpakov99k]; the number of occurrence of the tandem repeat with each given copy exponent; the shortest tandem repeat that begins at each position in the sequence [Kosaraju94], etc.

We also introduce the notion of primitive tandem repeat family, (***PTR family***, for short). For a given primitive tandem repeat (PTR), the PTR family is a set with one entry for **each different copy exponent** of the PTR. Each entry in the PTR family includes the copy exponent, the number of times each copy exponent occurred in the sequence, and the position of the first occurrence with respect to the sorted suffixes. Members of a PTR family will all share the same base primitive sequence. Thus, for each PTR, we can have at most one family. Using the BSCP data structure, we develop a linear time algorithm to identify all the distinct tandem repeats, and their respective PTR families. Although we can find the distinct PTRs and their families in $O(u)$ time, where u = length of the input sequence, reporting all the $\eta_{occ} = \sum_{\beta,l} \eta_{\beta,l}$ occurrences of the tandem repeats will still require an $O(u \log u)$time, since it is known that $\eta_{occ} = O(u \log u)$ [Crochemore81, Apostolico83p]. Here, $\eta_{\beta,l}$ = number of occurrence of PTR β with copy exponent l.

Compared with previous work in this area, our work is more closely related to methods that are based on suffix trees [Apostolico83p, Gusfield97, Stoye98g, Stoye2002g], or suffix arrays [Frankel2002st]. Stoye and Gusfield focused on syntactically distinct primitive tandem repeats, or essentially on determining the vocabulary of primitive tandem repeats. For each PTR, there is one entry in the PTR vocabulary, independent of the number of the copy exponents, or the number of ***distinct*** exponents. For some biological problems however, just the vocabulary of repeats is not enough. Examples here include in studying the role of tandem repeats in certain genetic diseases such as Huntington's disease, or Fragile X mental syndrome [Bat97ka,

---

* Authors are with the Lane Department of Computer Science and Electrical Engineering, West Virginia University, Morgantown, WV 26506. email: [don, feng]@csee.wvu.edu. This work was partially supported by a DOE CAREER Grant, No.: DE-FG02-02ER25541.

IEEE COMPUTER SOCIETY

Sinden2002popls], or in studying individual variability in genetic profiling [Jefreys85wt, Nakamura87etal]. In such environments, the specific copy exponents for the PTRs, and the number of such copy exponents are critical parameters for consideration. Various software tools have also been developed for the analysis of different forms of repetitions in a genomic sequence. For examples see [Kolpakov2003bk, Benson94w, Volfonsky2001hs, Lefebvre2003lda].

In the next section, we give a brief description of the SCP data structure, and some of its properties. This motivates the proposed block SCP data structure, and its construction, as described in Section 3. Section 4 presents the BSCP tree, and a method for locating distinct primitive tandem repeats (PTRs) and their respective families using the BSCP tree. Section 5 gives a brief complexity analysis for our method. The paper is concluded in Section 6.

2. The SCP Data Structure

Let T be an input sequence of length $u = |T|$. Suppose we pre-compute the longest-common-prefix (LCP) between *all* pairs of the sorted suffixes from a sequence, T. Using the relationship between the BWT [Burrows94w] and the suffix tree, in addition to the auxiliary arrays previously described in [Adjeroh2002zmpb, Bell2002pbma], we can obtain theses sorted suffixes directly from the compressed sequence. We store this information in a table. Since the table is based on the sorted suffixes, we call this the **sorted common prefix (SCP)**. Although the SCP and traditional LCP store the same basic information, the structure of the SCP is completely different. Also, the sorted nature of the suffixes for the SCP will have implications in the computation of this table, and its diverse uses. Given the i-th and j-th sorted suffixes, (SS_i and SS_j respectively), if SCP(i,j)=k, it means that the first k positions in the i-th suffix and the j-th suffix are exact matches (i.e. $SS_i[1..k] = SS_j[1..k]$, and $SS_i[k+1] \neq SS_j[k+1]$). The SCP is the starting point for our approach.

2.1 Nature of the SCP Data Structure

An example SCP table for a short sequence is given in Fig. 2. The SCP provides a cluster of similar areas in the sequence. Compared with the LCP, the structural nature of the SCP provides more explicit information about the nature of the original sequence. More importantly, this structure can be used in efficient applications of the SCP, and its construction. We observe the following properties from the SCP structure. Let i,j,k, be indices for the sorted suffixes, such that $i<j$, and $j<k$. Then, $SCP(i,j) \geq SCP(i,k), \forall k > j$. Further,

if $SCP(i,j) \geq 0$ and $SCP(i,k) = 0$,

then $\forall k > j$, $SCP(j,k) = 0$.

More generally, let $x = SCP(i,k)$.

Then $\forall k > j$, $SCP(j,k) = SCP(i,k)$

$$+ SCP(SS_j[x+1,...,u], SS_k[x+1,...,u])$$

$$= x + SCP(SS_j[x+1,...,u], SS_k[x+1,...,u]).$$

The SCP is symmetric: SCP(j,k)= SCP(k,j). It is also usually sparse, although not always. Example, with T = *AAAAA* will have a full SCP table, where the row entries are of the form **1 1 1 1**; **2 2 2**; **3 3**; **4**.

2.2 Computing the SCP

The computation of the SCP is performed based on some auxiliary data structures from the output of the Burrows Wheeler Transform, BWT [Burrows94w]. In order to provide some form of random access to the transformed BWT output, we introduced auxiliary transformation vectors [Adjeroh2002mpbz, Bell2002pbma]. Define an array *Hr* computed from the BWT *V* array by the following algorithm: *x:=id; for i:=1 to u do* {*x*:=*V*[*x*];*Hr*[*u*+*1*-*i*]:=*x*}. Given the arrays *F* and *Hr*, the original text can be retrieved by applying the *inverse* transformation $T[i]= F[Hr[i]], 1 \leq i \leq u$. Another auxiliary array used is *Hrs*, defined as the inverse of *Hr*. That is, $T[Hrs[i]] = F[i] = F[Hr[Hrs[i]]]$. As an example, with T=ACTAGA, *L*=GATAAC, *id=2*, *F*=AAACGT, *V* = [5 1 6 2 3 4], *Hr*=[2 4 6 3 5 1] and *Hrs*=[6 1 4 2 5 3]. These auxiliary data structures can be computed in O(u) time, from the BWT output [Adjeroh2002zmpb, Bell2002pbma]. In [Adjeroh2003f], three algorithms were proposed to compute the SCP, based on the auxiliary arrays. With the last algorithm, given a string $T = t_1 t_2 ... t_u$, and an equi-probable symbol alphabet Σ, the table of sorted common prefixes (SCP) can be computed using $O(u + |\Sigma| \kappa_{\max})$ number of comparisons on average, and $O(u|\Sigma|)$ worst case, with an extra space in $O(\kappa_{\max})$ on average, and $O(u)$ worst case, where $\kappa_{\max}$ is the maximum SCP value. Thus, for a fixed Σ, the SCP can be computed using an $O(u)$ number of comparisons.

3. The Block SCP

When the objective is to find tandem repeats, we can construct a more simplified structure called the ***Block Sorted Common Prefix***, (BSCP for short). This is motivated by the nature of the SCP. Unlike the SCP table, the BSCP stores only one value for blocks of suffixes in the SCP. Thus, to represent the BSCP, all we need is an indication of the starting and ending sorted suffixes for the block, and the SCP value at the block (i.e. the length of the common prefix for the suffixes in the block. Alternatively, we can store the position of the first

IEEE COMPUTER SOCIETY

mismatch for the suffixes). An example BSCP table is given in Fig. 1. The major advantage here is that the storage for the BSCP is much more reduced when compared with the storage requirement for the SCP. Also, the BSCP could be potentially faster to build than the complete SCP table. We observe that, although the BSCP is motivated by the SCP structure, we do not need to construct the SCP before we construct the BSCP. The BSCP can be obtained directly from the F-array, using the auxiliary arrays used to construct the SCP table.

3.1 Properties of the BSCP

We describe some important properties of the BSCP in the form of two lemmas, without proofs. Let the mismatch point m split the SCP table into two regions: top region with sorted suffix indices in the range $[s\ \ m]$, and bottom region with indices in the range $[m+1,\ e]$ in F. Here, s is the index to the starting suffix, and e is the index to the ending sorted suffix. That is, m is the index of the median suffix SS_{med} in the block, where we define the median as follows: $med = m = \min_i \text{ st. } SCP(i,b) > SCP(s,b)\ \forall i, s < i < b$.

In general, the i-th sorted suffix is obtained from the auxiliary arrays as follows:

$$SS_i = \begin{pmatrix} F[Hr[Hrs[i]]] \circ F[Hr[Hrs[i]+1]] \circ \\ F[Hr[Hrs[i]+2]] \circ \cdots \circ F[Hr[u]] \end{pmatrix},$$

where "$\circ$" is the concatenation operation. Hence, the median suffix will be given by

$$SS_{med} = \begin{pmatrix} F[Hr[Hrs[m]]] \circ \\ F[Hr[Hrs[m]+1]] \circ \cdots \circ F[Hr[u]] \end{pmatrix}.$$

Lemma 1: Median Suffix. *Let $a \pm b$ denote* $a+b$ or a-b. *Let the median suffix* SS_m (i.e. *mismatch point* m) *split the SCP table into two regions: top region with sorted suffix indices in the range* $[s\ m]$, *and bottom region with indices in the range* $[m+1,\ e]$ *in F. Let β be the longest common prefix between SS_s and SS_e. That is, $|\beta| = SCP(s,e)$. Then, β is a primitive tandem repeat with period $\delta = |\beta|$ in the original sequence T, if and only if:*

$$Hr[Hrs[m+1] \pm \delta] \in [s,m]\ \vee$$
$$Hr[Hrs[m] \pm \delta] \in [m+1,e].$$

Suppose the original sequence contains distinct primitive tandem repeats (PTR) with different repeating patterns, and each PTR can occur multiple times in the sequence, at potentially different locations. A primitive tandem repeat could be a prefix of another primitive tandem repeat. Then, by the sorted nature of the SCP data structure, the different copies of each different tandem repeat will cluster in different regions of the SCP table. This is based on the fact that each PTR partitions the SCP table into blocks. This motivates the following lemma:

Lemma 2: Uniqueness of the BSCP. *Given the BSCP table for an input sequence T, each SCP block has a potential to identify at most one unique PTR in T.*

The above implies that some SCP blocks may not necessarily correspond to any PTR in the sequence. Hence, the number of blocks in the BSCP can be no less than the number of distinct primitive tandem repeats. The lemma also guarantees that we can find ***all*** the primitive tandem repeats using the BSCP block structure.

4. The BSCP Tree

An immediate observation on BSCP is that we can represent the information in the BSCP table in a tree structure. Here, each block in the table will form one node in the tree. We call this the BSCP tree. Essentially, the BSCP tree is a binary tree, whereby the root node is the common prefix shared by all the suffixes in the set. For each σ-partition ($\sigma \in \Sigma$), we construct a BSCP tree, λ_σ. By combining the individual trees from each σ-partition, we can build λ_T, the BSCP tree for the entire sequence, T. Fig. 2 is a complete example, showing the SCP table, BSCP table, and BSCP tree for a sample sequence.

4.1 The PTR Family

For a given distinct primitive tandem repeat (PTR), the PTR can occur at various locations in the sequence T, and with different copy exponents. For each distinct PTR in T, we define the ***PTR family*** as the set of occurrences of the PTR with distinct copy exponents. For each PTR, β the PTR family contains one entry for each occurrence of β with a different exponent l; $\eta_{\beta,l}$ the number of times β occurred with this exponent; and $t_{\beta,l}$ the position of the first occurrence of β with the given exponent l. (Here, position is with respect to the F array). That is, members of a given PTR family all share the same base primitive, β. The notion of PTR family is motivated by problems in biology, such as analysis of susceptibility to certain genetic diseases, where the occurrence of a tandem repeat with a particular exponent, or how many times a PTR occurs with a given exponent is critical in analyzing the biological sequence.

Let $n_1 \to n_2 \to n_3 \to \cdots \to n_p$ denote a sub-path in the BSCP tree λ_T, such that, n_1 is the node with the PTR β (and hence lowest copy exponent), n_2 is the node with the next lowest copy exponent, etc., while n_p is the node with the maximum copy exponent. For a given PTR family, with base string β, we call this sub-path its ***family-path.*** The following establishes the uniqueness of the PTR family-path.

IEEE COMPUTER SOCIETY

Lemma 3: Uniqueness of the family-path. *Given the BSCP tree* λ_T*, the PTR families partition* λ_T*, such that for each PTR family, there is a sub-path in* λ_T *(called* family-path*) that is shared by only members of the family. No two PTR families share the same family-path.*

For a given family-path, starting from the first node, n_1, all the subsequent nodes will be either left-left nodes, or right-right nodes. That is, after identifying the first node n_1 in the family path, if n_2 is the left child of n_1, then n_3 must be a left child of n_2, n_4 must be a left child of n_3, etc. This is due to the sorted nature of the suffixes used to construct the BSCP data structure. We exploit this left-left or right-right property of the BSCP tree in locating members of a PTR family, after the base PTR has been found. It also provides an avenue for a faster search for the distinct primitive tandem repeats.

4.2. Locating PTRs with the BSCP Data Structure.

Given the BSCP data structure, we can locate tandem repeats by simply traversing the BSCP tree, and then checking if the conditions for tandem repeats are met by the common prefix stored at each node in the tree (Lemma 1). When we identify a common prefix that represents a primitive tandem repeat, we then determine the end point of the family path for the tandem repeats that belong to the same family as the current PTR. By doing this, we implicitly identify all the repetition exponents for the PTR, and the number of times the PTR appeared with each copy exponent. Based on the uniqueness lemma above, these copy exponents will all lie along the family-path. Using simple array indirections, we determine the location of the first occurrence (with respect to the *F*-array) of each copy exponent in the original sequence *T*. For PTRs that occurred with only two copies in the entire sequence, we use a special condition to identify them.

5. Complexity Analysis

Here, we state the space and time complexity of the approach described. With the BSCP structure, the analysis is simple, and some of the complexity results are straightforward. We provide pointers to the proofs, and omit detailed descriptions for brevity. A casual look at Fig.1 or Fig.2 shows that, on average, the number of blocks in the BSCP table (or number of nodes in the BSCP tree) will be linear with respect to the size of the input sequence.

Lemma 4. *Given the F, Hr and Hrs arrays for an input sequence T, the time required to construct the BSCP data structure (i.e. BSCP table) is in O(u), where u is the size of the original sequence. The space requirement for the BSCP data structure is in O(u).*

The time required to construct the BSCP table can be split into two parts – time needed for "horizontal matching" to determine the SCP values, and time needed for "vertical matching" to determine the **median suffix** (or the split point) using binary search at the current horizontal mismatch point. For the first part, the time required is in O(*u*), since a maximum of *u* horizontal comparisons will be made. For the second part, at each recursion step, a smaller vertical block is considered, and the search for the median block is performed on this smaller block using binary search. It can be shown that, for each σ-partition, the total number of comparisons performed by this recursive procedure cannot be greater than: $c(v) = \log v + 2\log\frac{v}{2} + 4\log\frac{v}{4} + ... + 2^i \log\frac{v}{2^i}$, where $i = \log v - 1$, and $v = \frac{u}{|\Sigma|}$ is the size of the σ-partition, on average. Thus, $c(v) = 2v - \log v - 2$. Or, $c(v) = O(v) = O\left(\frac{u}{|\Sigma|}\right)$. The space complexity follows easily since we have O(*u*) entries in the table.

Theorem 1. *Given the F, Hr and Hrs arrays for an input sequence T, the time required to construct the BSCP tree* λ_T *is in O(u), where u is the size of the original sequence. The space requirement for the BSCP tree is in O(u).*

We can either construct the BSCP tree by first constructing the BSCP table, or directly from the *F* and auxiliary arrays. Given the BSCP table, we see that at each node in the BSCP tree, the only work that is required is to provide pointers to the left and right nodes, and to insert the SCP value, the current node index number, and the start and end suffixes that define the block. We do this recursively at each node, until we have considered all the nodes in the tree. Since the total number of nodes in the BSCP is linear with respect to the length of the sequence, the theorem follows.

Theorem 2. *Given the BSCP tree* λ_T *for an input sequence T, we can locate all the PTRs and the PTR families in T in O(u) time and O(u) space.*

Using the BSCP tree to locate PTRs basically involves traversing the tree. The processing at each node during this traversal simply implements the conditions for PTRs using the BSCP table. Thus the correctness of the approach is based on Lemma 1 – on the median suffix. The proof of the complexity bounds is based on the uniqueness lemmas (Lemma 2 and Lemma 3), and the fact that the BSCP tree can contain at most *u* nodes for a *u*-length sequence. To search for tandem repeats, we do not need to process all the nodes in the tree. For the nodes that need to be processed, we need to process them at most once. The processing at each node requires a constant time – basically a constant number of

IEEE COMPUTER SOCIETY

comparisons using the bounds on the block suffixes – i.e. the starting and ending sorted suffixes. After constructing the BSCP tree, we do not need any extra space to find the tandem repeats, beyond those required for the auxiliary arrays.

6. Conclusion

In this paper, we introduce a novel data structure, called the BSCP, which can be constructed in O(u) time and space, where u is the length of the input sequence. We also introduce the notion of *PTR family* – a biologically motivated description and representation of the tandem repetitions in a sequence. Based on the BSCP tree, we present an algorithm to locate all the primitive tandem repeats (PTR), and the PTR family for each primitive tandem repeat in an input sequence T. The algorithm requires O(u) time and space on average. Our methods are based on the auxiliary arrays produced during the encoding and decoding stages of the Burrows Wheeler Transform.

References

[Adjeroh2003f] Adjeroh D.A., and Feng J. "The SCP and compressed domain analysis of biological sequences", *Proc., IEEE Bioinformatics Conference*, August 2003.

[Adjeroh2002zmpb] Adjeroh D.A., Zhang Y, Mukherjee A., Powell M., and Bell, T.C. "DNA sequence compression using the Burrows-Wheeler Transform", *Proc., IEEE Bioinformatics Conference*, August 2003, pp. 303-313, 2002 .

[Apostolico 83p] Apostolico A, Preparata F. P, "Optimal off-line detection of repetitions in a string", *J. Theoretical Computer Science, 22: 297-315* ,1983.

[Bat97ka] Bat O, Kimmel M., and Axelrod D. E. "Computer simulation of expansions of DNA triplet repeats in the Fragile X Syndrome and Huntington's disease", *Journal of Theoretical Biology*, 188, 53-67, 1997.

[Bell2002pma] Bell T, Powell M, Mukherjee A and Adjeroh D. "Searching BWT compressed text with the Boyer-Moore algorithm and binary search", *Proc. IEEE Data Compression Conference*, Snowbird, Utah, 2002.

[Benson94w] Benson G. and Waterman M.S, "A method for fast database search for all *k*-nucleotide repeats", *Nucleic Acid Research*, 22, 4828-4836, 1994.

[Burrows94w] Burrows M. and Wheeler D.J, "A block-sorting lossless data compression algorithm", *Technical Report*, Digital Equipment Corporation, Palo Alto, CA, 1994 .

[Crochemore81] Crochrmore M. "An optimal algorithm for computing repetitions in a word", *Information Processing Letters*, 12, 244-250,1981.

[Franek03st] Franek F, Smyth W. F and Tang Y. "Computing all repeats using suffix arrays", *J. Automata, Languages & Combinatorics* 8-4, 579-591,2003.

[Gusfield97] Gusfield, D. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*, Cambridge University Press, UK, 1997.

[Jeffreys85wt] Jeffreys AJ, Wilson V., and Thein SL, "Individual-specific fingerprints of human DNA", *Nature*, 316(6023):76-9, 1985.

[Kolpakov 99k] Kolpakov R.M., Kucherov G., "Finding maximal repetitions in a word in linear time", *FOCS*, 596-604, 1999.

[Kolpakov03bk] Kolpakov R.M., Bana G., Kucherov G., "mreps: Efficient and flexible detection of tandem repeats in DNA", *Nucleic Acids Research 31*(13): 3672-3678, 2003.

[Kosaraju94] Kosaraju S. R. "Computation of squares in a string", *CPM*: 146-150, 1994.

[Lefebvre03da] Lefebvre A., Lecroq T., Dauchel H. and.Alexandre, J., "FORRespeats: detects repeats on entire chromosomes and between genomes", *BioInformatics* 19(3):319-326,2003.

[Majewski2000o] Majewski J and Otto J, "GT repeats are associated with recombination on human chromosome 22", *Genome Research*. 10(8):1108-14, 2000.

[Nakamura87etal] Nakamura Y., Leppert M., O'Connell P., Wolff R, Holm T., Culver M., Martin C., Fujimoto E., Hoff M., Kumlin E. and White R., "Variable number of tandem repeat (VNTR) markers for human gene mapping", *Science* 235:1616-1622, 1987.

[Richards93hys] Richards RI, Holman K, Yu S and Sutherland GR, **"**Fragile X syndrome unstable element, *p*(CCG)*n*, and other simple tandem repeat sequences are binding sites for specific nuclear proteins", *Human Molecular Genetics*, 2, 1429-1435, 1993.

[Sinden02popls] Sinden R.R, Potaman V. N, Oussatcheva E. A, Pearson C.E, Lyumchenko Y. and Shlyakhtenko L. S, "Triplet repeat DNA structures and human genetic disease: dynamic mutations from dynamic DNA", *Journal of Biosciences. 24(1): 53-65,* 2002.

[Stoye98g] Stoye J. and Gusfield D, "Simple and flexible detection of contiguous repeats using a suffix tree", Proceedings, *CPM*: 140-152, 1998.

[Stoye2002g] Stoye J. and Gusfield D, "Simple and flexible detection of contiguous repeats using a suffix tree", *J. Theoretical Computer Science*, 270(1-2),843-856, 2002.

[Volfovsky2001hs] Volfovsky N. Hass B.J., Salzberg S, "A clustering method for repeat analysis in DNA sequences", *Genome Biology*, **2**(8), 2001.

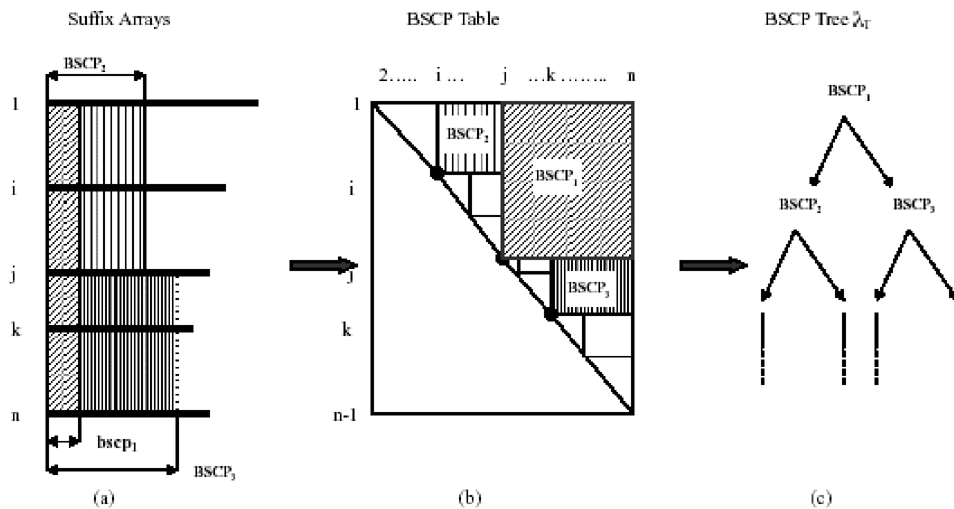

Fig.1: Schematic diagram for BSCP and BSCP Tree: (a) sorted suffixes, (b) the BSCP table, (c) the BSCP Tree.

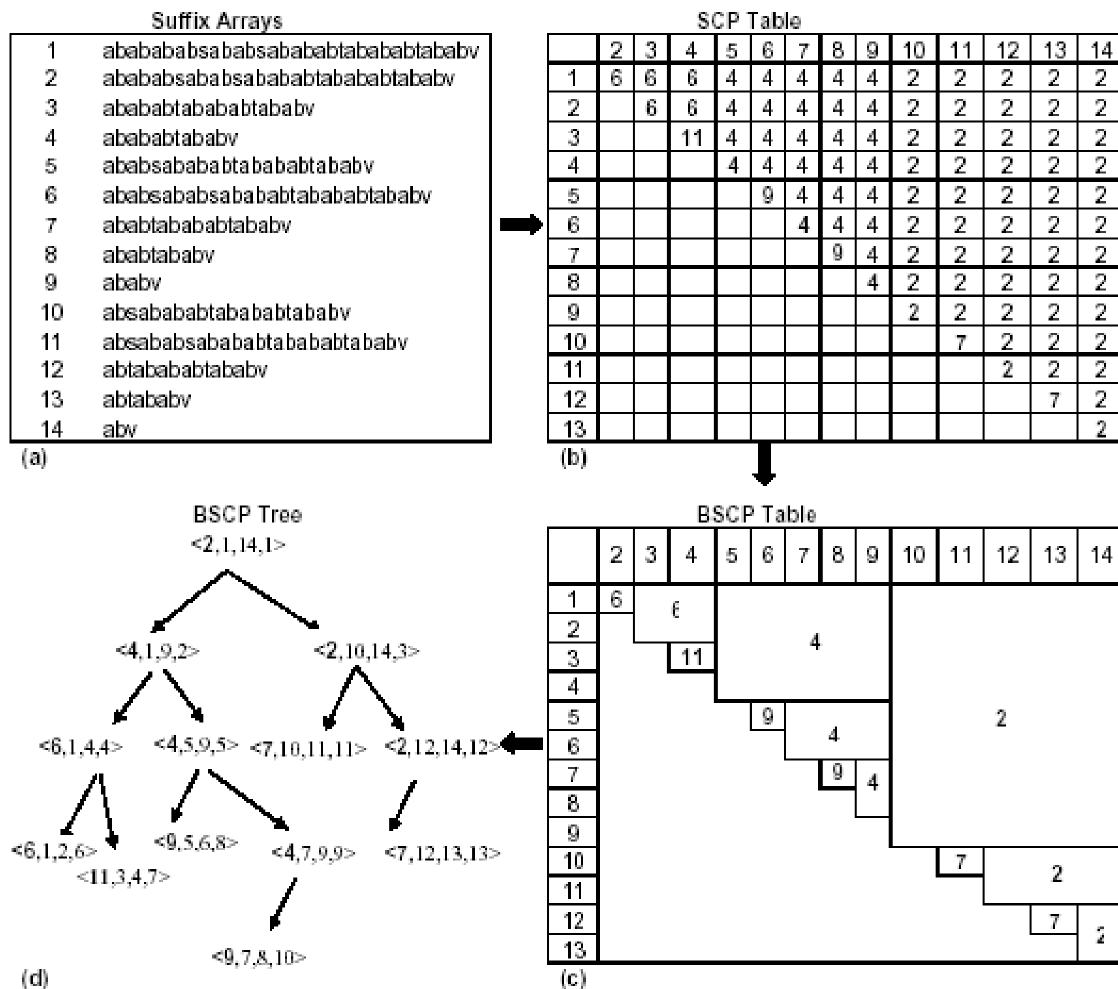

Fig. 2: (a) Suffix arrays, (b) SCP table, (c) BSCP table, and (d) BSCP tree for a sample sequence *T=abababababsababsabababtabababtababv*. In (c), numbers inside the box represent the block SCP values. In (d) each node is represented with the 4-tupple <*k*, s, e, *index*> where *k* is the block SCP value, *s* is index of the starting sorted suffix, *e* is the index of the ending suffix, and *index* is the block label.

IEEE COMPUTER SOCIETY