BIOINFORMATICS APPLICATIONS NOTE Vol. 15 no. 5 1999
Pages 426–427

REPuter: fast computation of maximal repeats in complete genomes

Stefan Kurtz and Chris Schleiermacher

Technische Fakultät, Universität Bielefeld, Postfach 10 01 31, D-33501 Bielefeld, Germany

Received on January 11, 1999; revised on February 22, 1999; accepted on February 23, 1999

Abstract

Summary: *A software tool was implemented that computes exact repeats and palindromes in entire genomes very efficiently.*
Availability: *Via the Bielefeld Bioinformatics Server (http://bibiserv.techfak.uni-bielefeld.de/reputer/).*
Contact: *{kurtz,icschlei}@techfak.uni-bielefeld.de*

Introduction

Computer scientists have developed methods to locate repeated substrings of different kinds, dating back to the pioneering work of Martinez (1983). Several software tools have been developed to locate repeated substrings, for example Devereux *et al.* (1984), Agarwal and States (1994) and Rivals *et al.* (1997). However, to our knowledge all available software tools for repeat analysis have strict limits on the maximal length of the input sequence they allow to process. For example, the repeatfinder of the GCG-package (version 7.0) only allows input sequences of length up to 350 000 bases.

We have developed a software tool *REPuter* which allows to determine all exact repetitive substrings contained in complete genomes. Exact repeats are only a small fraction of all repeats of biological interest. However, since exact repeats usually form the core blocks of approximate repeats, *REPuter* can also serve as a fast subroutine for programs that detect these, see for example Leung *et al.* (1991). The running time and space requirement of *REPuter* is linear in the length of the genome and in the size of the output. The method we use is thus asymtotically optimal. The main advantage over previous programs is the reduced time and space complexity of our tool, and the guaranteed linear worst case behavior. This allows one to handle much longer input sequences. The current version is able to process input sequences consisting of up to 67 million bases. For example, to compute all maximal repeats of length at least 20 contained in the *S. cerevisiae* genome (12 147 818 bases) takes about 46 s on a Pentium 350 MHz-computer, using 160 Mbyte of space.

REPuter consists of two programs, a search engine and a visualizing component. The search engine processes a DNA sequence given by the user in Fasta-format, and returns a representation of all maximal repeats in a simple ASCII-format. The visualizing component processes the output of the search engine and produces an overview of the number, the length, and the location of the repeated substrings. *REPuter* is available via the WWW on the Bielefeld-Bioinformatics Server (see above). For user convenience we have precomputed the maximal repeats for some genomes. The search engine can be downloaded as an executable binary for several platforms.

Methods

Consider some sequence s over the DNA alphabet. A *repeat* is a substring in s which occurs at least twice. Suppose w is a repeat of length l in s which occurs at the starting positions i and j, i.e. $s_i \ldots s_{i+l-1} = s_j \ldots s_{j+l-1}$. Let $i < j$. Then we say that (l,i,j) is a *forward repeat* of length l, F-repeat, for short. Note that w is contained in a longer repeat if the bases to the left or to the right of both occurrences of w are identical. To reduce redundancy, we restrict attention to maximal F-repeats: (l,i,j) is *maximal* if $(s_{i-1} \neq s_{j-1})$ and $(s_{i+1} \neq s_{j+l})$ (whenever these positions exist in s). Note that each F-repeat in s can easily be deduced from a maximal F-repeat. Maximal palindromes are defined in a similar way: (l,i,j), $i \leq j$, is a *palindromic repeat*, P-repeat, for short, if $s_i \ldots s_{i+l-1} = \overline{s_j \ldots s_{j+l-1}}$, where $\overline{w}$ denotes the reverse complement of a DNA-sequence w. (l,i,j) is *maximal* if the complement of base s_{i-1} and s_{j-1} is different from s_{j+1} and s_{i+1}, respectively, whenever these positions exist in s.

Example 1

gtcaca contains one maximal F-repeat (2,2,4) (*ca*) and one maximal P-repeat (2,0,3) (*gt*), of length ≥2.

For a given threshold $l > 0$, our search engine computes all maximal repeats of length at least l, using a variation of an algorithm described in Gusfield (1997). Two phases are necessary to deliver all maximal F- and P-repeats for a sequence s:

In the first phase, the suffix tree for the sequence $t = xsy\overline{s}z$ is computed. x, y, and z are unique symbols not occurring in s. They allow to conveniently handle boundary cases. The suffix tree is a well known data structure in string processing (McCreight, 1976). It can be computed in $O(n)$ time and space, where n is the length of t. The tool of Rivals *et al.* (1997) is also based on suffix trees, but it computes non-overlapping F-repeats, and the worst case running time is quadratic.

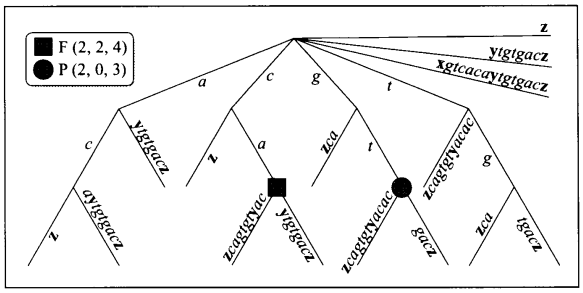

Fig. 1. The suffix tree for t = *xgtcacaytgtgacz* Marked nodes represent maximal repeats.

In the second phase, a depth first traversal of the suffix tree locates all nodes in the tree representing maximal repeats and delivers their lengths and starting positions. The traversal requires $O(n)$ time, and all maximal repeats are output in $O(m)$ time where m is their number.

The running time of our program is $O(n + m)$ which is optimal. Using the space reduction techniques of Kurtz (1998), the program requires about 12.5n bytes for representing the suffix tree of t. This is an improvement of 7n bytes over previous implementation techniques, see for example McCreight (1976). Let $r_{\max}$ be the maximal length of any repeated substring in t. For any sequence w let $p(w)$ be the number of positions in t where w occurs, and let $p_{\max}$ be the maximal $p(w)$ over all words w of length at least l. The space requirement for the traversal phase is $4\,p_{\max} + 45r_{\max}$ bytes which is independent of the size of the output.

Example 2

Figure 1 shows the suffix tree for the sequence t = *xgtcacaytgtgacz*. The maximal F- and P-repeats of length at least 2 contained in the sequence *gtcaca* (see Example 1) can be read from the two marked branching nodes. All other branching nodes either represent repeats which are too short (see *a*,*c*,*g* and *t*), or they represent repeats which are equivalent to one of the reported maximal repeats (*tg*/*tg* is equivalent to (2,2,4) and *ac*/*ac* is equivalent to (2,0,2)).

Table 1 shows the running times and space requirements for computing all maximal F- and P-repeats of length at least 20 for several genomes (SUN Enterprise, 166 MHz, 512 Mbyte RAM, and Pentium II, 350 MHz, 384 Mbyte RAM).

We have developed software to visualize the lengths of the repeats and their location in the form of a *repeat graph*. The repeat graph consists of two thick horizontal lines. The line on the top represents the input sequence s. The line on the bottom represents a copy of s or the reverse complement s, depending on whether to visualize maximal F-repeats or maximal P-repeats. Each thin line connects a maximal repeat, which is represented as two colored blocks on both horizontal lines. Different colors stand for repeats of different lengths. Figure 2 shows the repeat graph for a sample sequence.

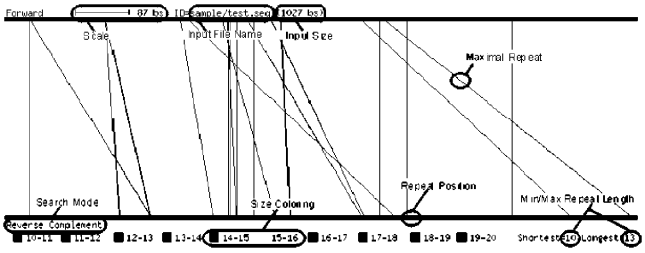

Fig. 2. *REPuter* graphical output: repeat graph for a sample sequence.

Table 1. Results for F- and P-repeats of length ≥20. The time and space for computing only the F-repeats is smaller by a factor of 1/2, since this task requires to compute the suffix tree for t = *xsz*

Input, approx. size (MB)	F-repeats Num. of repeats	SUN (s)	PII (s)	Mem. (MB)	P-repeats Num. of repeats	SUN (s)	PII (s)	Mem. (MB)
M. gen, 0.6	1022	4	2	8	56	9	4	15
C. tra, 1.0	26	8	3	14	8	17	7	27
A. ful, 2.1	4197	18	7	29	3267	39	16	57
E.coli, 4.6	7799	40	17	61	6763	83	35	122
S. cer, 11.58	173 526	112	46	160	171 841	248	97	320

Acknowledgments

We would like to thank Christoph Sensen who suggested the development of a tool for searching repeats. Philip Denno gave hints to improve the output and Robert Giegerich suggested the name for the tool. Stefan Kurtz was supported by DFG grant Ku 1257/1-1 and Chris Schleiermacher by the Boehringer Ingelheim Fonds.

References

Agarwal,P. and States,D. (1994) The repeat pattern toolkit (rpt): Analyzing the structure and evolution of the c. elegans genome. *Proc. ISMB '94*, pp. 1–9.

Devereux,J., Haeberli,P. and Smithies,O. (1984) A comprehensive set of sequence analysis programs for the VAX. *Nucleic Acids Res.*, **12,** 387–395.

Gusfield,D. (1997) *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, New York.

Kurtz,S. (1998) Reducing the space requirement of suffix trees. Report 98–03, Technische Fakultät, Universität Bielefeld.

Leung,M., Blaisdell,B., Burge,C. and Karlin,S. (1991) An efficient algorithm for identifying matches with errors in multiple long molecular sequences. *J. Mol. Biol.*, **221,** 1367–1378.

Martinez,H. (1983) An efficient method for finding repeats in molecular sequences. *Nucleic Acids Res.*, **11,** 4629–4634.

McCreight,E. (1976) A space-economical suffix tree construction algorithm. *JACM*, **23,** 262–272.

Rivals,É., Dauchet,M., Delahaye,J. and Delgrange,O. (1997) Fast discerning repeats in DNA sequences with a compression algorithm. *Proceedings of the Workshop on Genome and Informatics, Tokyo*.