Bioinformatics, 32(17), 2016, 2707–2709
doi: 10.1093/bioinformatics/btw298
Advance Access Publication Date: 11 May 2016
Applications Note

Sequence analysis

SA-SSR: a suffix array-based algorithm for exhaustive and efficient SSR discovery in large genetic sequences

B. D. Pickett, S. M. Karlinsey, C. E. Penrod, M. J. Cormier, M. T. W. Ebbert, D. K. Shiozawa, C. J. Whipple and P. G. Ridge*

Department of Biology, Brigham Young University, Provo, UT 84602, USA

*To whom correspondence should be addressed.

Associate Editor: John Hancock

Received on February 10, 2016; revised on April 21, 2016; accepted on May 4, 2016

Abstract

Summary: Simple Sequence Repeats (SSRs) are used to address a variety of research questions in a variety of fields (e.g. population genetics, phylogenetics, forensics, etc.), due to their high mutability within and between species. Here, we present an innovative algorithm, SA-SSR, based on suffix and longest common prefix arrays for efficiently detecting SSRs in large sets of sequences. Existing SSR detection applications are hampered by one or more limitations (i.e. speed, accuracy, ease-of-use, etc.). Our algorithm addresses these challenges while being the most comprehensive and correct SSR detection software available. SA-SSR is 100% accurate and detected >1000 more SSRs than the second best algorithm, while offering greater control to the user than any existing software.
Availability and implementation: SA-SSR is freely available at http://github.com/ridgelab/SA-SSR
Contact: perry.ridge@byu.edu
Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Simple Sequence Repeats (SSRs), microsatellites, or short tandem repeats (STRs), are tandem repeats of short (often 2–5 bp) nucleotide strings (Madesis *et al.*, 2013). There are generally 10–100 such repeats at each SSR locus resulting in a DNA segment that is amenable to rapid molecular characterization. Given their repetitive nature, the lengths of SSR loci tend to increase or decrease due to polymerase slippage during DNA replication (Schlotterer and Tautz, 1992). As a consequence, SSR loci have high mutation rates and frequently generate multiple polymorphic alleles. SSR loci are common in both nuclear and organellar genomes, and when flanked by unique sequence, PCR primers can be readily designed to amplify simple sequence length polymorphisms. SSRs have proven highly useful for a variety of molecular genetic, population genetic and phylogenetic applications because it is simple to genotype them using PCR, and because they are highly polymorphic.

While SSRs have been extensively characterized in many model species, the expense and effort traditionally required to develop SSRs has limited their use in non-model species. Fortunately, next-generation sequencing has enabled researchers to quickly produce large quantities of genomic and/or transcriptomic data for nearly any species. While a high quality genome is still difficult to assemble, there is usually adequate sequence information to identify thousands of unique SSR loci with minimal sequencing. Thus, researchers working in non-model systems need user friendly and customizable bioinformatics algorithms to identify SSR loci.

A complete, accurate, characterization of SSRs in non-model systems increases the likelihood researchers are able to identify SSRs where flanking genotyping primers can be designed. SSR differences can be used to differentiate between related species or provide insights into specific phenotypes/adaptations. Finally, since the majority of researchers do not have formal computational training, a straightforward, intuitive application is likely to enable traditional bench/field scientists to use SSRs in their research.

Many tools exist to find SSRs with varying degrees of utility, but few tools have both a useful command line interface for scripting and meaningful, parseable output. Identifying SSRs in a sequence is challenging because the search is prohibitive in time and memory requirements. Most existing tools use either an exhaustive, combinatorial search approach or a heuristic approach (Lim *et al.*, 2013). Exhaustive searches have time complexity that grows exponentially, while heuristic approaches trade comprehensiveness for run time. We developed an algorithm that is both efficient and complete.

Conceptually, finding SSRs in a nucleotide sequence is relatively straightforward, but the size of current datasets makes it a substantial challenge. SSR detection in sequence data is a substring operation—a large class of problems common in computer science. Many algorithms and data structures have been developed to reduce the time and space requirements for string operations. The suffix tree boasts linear time and space requirements for generating its representation of the string and can be used to perform many important substring operations in O(n log n) time. After Weiner discovered suffix trees (Weiner, 1973), McCreight (McCreight, 1976) and Ukkonen (Ukkonen, 1995) each simplified it, paving the way for the development of the suffix array (Abouelhoda *et al.*, 2004; Kurtz, 1999; Manber and Myers, 1993). Suffix arrays have the same properties as suffix trees, but are as many as five times more memory efficient (Kurtz, 1999; Manber and Myers, 1993).

2 Algorithm

A suffix array is an array of character positions representing a list of all possible suffixes of a string, ordered lexicographically, and longest common prefix arrays are arrays of the lengths of the longest common prefix of each adjacent suffix in the suffix array. Using suffix and longest common prefix arrays, we designed and implemented a novel algorithm for finding SSRs in a nucleotide sequence in linear (O(n)) time and space. The algorithm makes no distinction between microsatellites or minisatellites—it can find tandem repeats of any length or period size.

SSRs are identified by calculating three different parameters, k, r and p from the suffix and longest common prefix arrays, where k equals the length of an SSR repeating unit or period size, r equals the number of times it repeats after the original occurrence, and p equals the position of the first nucleotide of the first period of the SSR (see Supplementary Texts 1 and 2, and Supplementary Figure S1 for a more detailed explanation). SSRs are identified by calculating k, p and r from the suffix and longest common prefix arrays (Supplementary Fig. S1C). Let i equal the index of any entry in the suffix array (except the first position), where SA and LCPA are the suffix and longest common prefix arrays, respectively:

$$k_i = |SA_i - SA_{i-1}| \quad (1)$$

$$r_i = \left\lfloor \frac{LCPA_i}{k_i} \right\rfloor \quad (2)$$

$$p_i = MIN(SA_{i-1}, SA_i) \quad (3)$$

If $r > 0$, an SSR of length $k * (r + 1)$ exists at position p in the original sequence, otherwise if $r = 0$ there is no SSR at position p. The base unit (e.g. AG in the SSR AGAGAG) of the SSR starts at position p and ends at position $p + (k - 1)$. Thus, by comparing each adjacent element in the suffix array we can find SSRs in a sequence.

3 Results

Our algorithm requires at most $9n$ bytes of memory, where n is the length of the entire query sequence. For each nucleotide in the sequence, we generously assume one byte in the original sequence (using 8-bit characters), four bytes in the suffix array (using 32-bit integers) and four bytes in the longest common prefix array (using 32-bit integers). The time complexity for building a suffix array and its longest common prefix array is O(n). Our algorithm then requires 3 * ($n - 1$) constant time computations to find SSRs, thus keeping the total time and space complexities at O(n).

We evaluated the performance of our algorithm compared to seven existing applications (see Supplementary Table S1 for a list of algorithms) on the *Arabidopsis thaliana (chromosome 4), Caenorhabditis elegans, Drosophila melanogaster, Escherichia coli and Zaire ebolavirus* genomes (GenBank Accessions: NC_003075.7, GCA_001483305.1, GCA_001014345.1, GCA_001432175.2 and NC_002549.1, respectively), comprised of 13 121 sequences totaling 248 846 830 nucleotides. Sequences ranged in length from 516 to 18 590 000 nucleotides with a median size of 4 662 (Supplementary Figures S2–S6 show a distribution of sequence lengths). Dozens of applications exist for SSR detection. We selected algorithms for comparison that: (i) were capable of processing the *Arabidopsis thaliana* chromosome (the longest of the sequences), (ii) had a non-interactive, Linux, command-line interface, (iii) were freely available for immediate download and (iv) had 10 or more citations per year or were published in the last three years. Several additional algorithms met our requirements, but used antiquated shared libraries, or had compile/run-time errors. All comparisons were run on a 6-core Intel Haswell Westmere (2.67 GHz) processor with 24 GB of memory (1066 MHz DDR3).

SA-SSR, like other algorithms, calls any detected sequence repeat an SSR. Reported numbers and accuracy reflect the assumption that all sequence repeats are SSRs. SA-SSR maximized the number of SSRs identified, while maintaining low memory requirements and runtime, and providing higher flexibility to the user to control desired output (results summarized in Table 1 with more detailed results in Supplementary Table S2). We counted the total number of SSRs identified by SA-SSR and each of the algorithms with period sizes one to seven and minimum total length of 16 nucleotides (period sizes and lengths likely to be of most interest in common applications). Next, we determined the accuracy of each of the tested algorithms, including SA-SSR, by writing a script to scan the entire sequence to verify whether or not a reported SSR was present. Most of the tested algorithms, including SA-SSR, were 100% accurate. However, compared to other algorithms, SA-SSR, found the highest number of correct (38 088 SSRs) and unique SSRs (on average >18 000 SSRs more than the other algorithms). MREPS, SSR-Pipeline and TRF only missed 1340, 3047 and 7423 correct SSRs detected by SA-SSR, respectively. However, TRF was only 23% accurate. Results of algorithm comparisons and software features are summarized in Supplementary Tables S2–S31.

Finally, we designed SA-SSR with intuitive features and formatting requirements. Like other SSR detection applications, SA-SSR takes FASTA files as input. However, some of the other applications, including some of those with high performance, are difficult to use. For example, MREPS displays an error message if any characters are not A, C, G, T or N, or if too many N's are present. Even if a user has the skills to remove all the characters that are not A, C, G or T, this makes the output positions of SSRs incorrect because those characters are not accounted for. Additionally, MREPS output is in a relatively un-structured text document that is not trivial to

Table 1. Summary of results from comparisons of SA-SSR with other SSR detection algorithms

	CPU time[a] (mm:ss)	Real time[a] (mm:ss)	SSRs reported	SSRs In range[b]	Number correct[c]	Percent correct	Comparison with SA-SSR: SSRs unique to software[d]	SSRs unique to SA-SSR	Shared SSRs
GMATo	329:18	329:18	72 713 858	15 284	6617	43.29	20	34 237	3851
MREPS	393:02	393:02	75 552	37 076	37 076	100	71	1340	36 748
PRoGeRF	3194:18	3194:18	5 457 129	2278	2268	99.56	2	35 864	2224
QDD	24:17	24:17	53 248	17 418	17 418	100	10	20 759	17 329
SA-SSR	28 820:12	2416:32	38 088	38 088	38 088	100	NA	NA	NA
SSR-Pipeline	1411:21	1411:21	60 344 067	36 398	36 398	100	68	3047	35 041
SSRIT	2:12	2:12	13 217	13 217	13 217	100	5	24 951	13 137
TRF	12:14	12:14	2 035 715	1 47 284	33 876	23.00	12	7423	30 665

This is a combination of results across each of the genomes included in the comparison. For more detailed results see Supplementary Tables S2, S4–S31.

[a]MREPS timing includes the pre- and post-processing time for each genome necessary to adjust positions to account for removing ‘incorrect symbols’ and Ns. The additional times are an average of multiple approaches.

[b]We only considered SSRs with period sizes 1–7 (inclusive) and lengths of at least 16 nucleotides (nt). The difference between the number of SSRs in range and reported is due exclusively to SSR length (less than 16 nt) and period size (greater than 7).

[c]Whenever possible, we salvaged correct SSRs that were inside incorrect SSRs reported by other software packages. For example, in *Drosophila melanogaster*, we recovered three for PRoGeRF and 8408 for TRF. To illustrate, in sequence JXOZ01000043.1, TRF reports a CT repeated 36 times at position 2171. While TRF does correctly identify a low-complexity region with many CT repeats, there are not 36 perfect repeats in a row. In this case, we salvaged two perfect CT regions, each repeating 8 times.

[d]Detailed pairwise comparisons can be found in Supplementary Tables S4–S31.

parse. As another example, SSR-Pipeline can only look for one period size at a time, requiring the user to manually re-run the software repeatedly for each period size of interest. Finally, SA-SSR provides greater flexibility to the user. For example, the user can choose whether to perform an exhaustive or faster (still nearly complete) search, change output filters to report (or not) overlapping SSRs, or report only user-specified SSRs.

SA-SSR is freely available at: http://github.com/ridgelab/SA-SSR.

Acknowledgements

We thank the Fulton Supercomputing Laboratory (https://marylou.byu.edu) at Brigham Young University for their consistent efforts to support our research.

Funding

This work was supported by start-up funds from Brigham Young University to PGR and a mentoring environment grant from Brigham Young University to CJW.

Conflict of Interest: none declared.

References

Abouelhoda,M.I. *et al*. (2004) Replacing suffix trees with enhanced suffix arrays. *J. Discrete Algorithms*, **2**, 53–86.

Kurtz,S. (1999) Reducing the space requirement of suffix trees. *Softw. Pract. Exp*., **29**, 1149–1171.

Lim,K.G. *et al*. (2013) Review of tandem repeat search tools: a systematic approach to evaluating algorithmic performance. *Brief. Bioinf*., **14**, 67–81.

Madesis,P. *et al*. (2013) Microsatellites: Evolution and contribution. In: Microsatellites. Springer, pp. 1–13.

Manber,U. and Myers,G. (1993) Suffix arrays: a new method for on-line string searches. *SIAM J. Comput*., **22**, 935–948.

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

Schlotterer,C. and Tautz,D. (1992) Slippage synthesis of simple sequence DNA. *Nucleic Acids Res*., **20**, 211–215.

Ukkonen,E. (1995) On-line construction of suffix trees. *Algorithmica*, **14**, 249–260.

Weiner,P. (1973) Linear pattern matching algorithms. Switching and Automata Theory, 1973. SWAT'08. In: IEEE Conference Record of 14th Annual Symposium on IEEE, pp. 1–11.