RAPID COMMUNICATION

SARP: A Novel Algorithm to Assess Compositional Biases in Protein Sequences

Kirill S. Antonets[1] and Anton A. Nizhnikov[1,2]

[1]Department of Genetics and Biotechnology, St. Petersburg State University, St. Petersburg, Russia. [2]St. Petersburg Branch of N.I. Vavilov Institute of General Genetics, Russian Academy of Sciences, St. Petersburg, Russia.
Corresponding author email: ant.nizhnikov@gmail.com

Abstract: The composition of a defined set of subunits (nucleotides, amino acids) is one of the key features of biological sequences. Compositional biases are local shifts in amino acid or nucleotide frequencies that can occur as an adaptation of an organism to an extreme ecological niche, or as the signature of a specific function or localization of the corresponding protein. The calculation of probability is a method for annotating compositional bias and providing accurate detection of biased subsequences. Here, we present a Sequence Analysis based on the Ranking of Probabilities (SARP), a novel algorithm for the annotation of compositional biases based on ranking subsequences by their probabilities. SARP provides the same accuracy as the previously published Lower Probability Subsequences (LPS) algorithm but performs at an approximately 230-fold faster rate. It can be recommended for use when working with large datasets to reduce the time and resources required.

Keywords: algorithm, protein, sequence analysis, probability, composition

Evolutionary Bioinformatics 2013:9 263–273

doi: 10.4137/EBO.S12299

This article is available from http://www.la-press.com.

Introduction

Compositional biases are local shifts in amino acid or nucleotide frequencies in biological sequences. This is a widespread natural phenomenon occurring at all levels of biological material, from genomes and proteomes down to short regions of genes and proteins. These regions are called compositionally biased (CB) regions. It is now clear that CB regions play a significant role in the adaptation of organisms to extreme ecological niches[1,2] and determine certain properties of proteins.[3,4] Some types of CB regions in protein sequences are strongly associated with completely disordered sequences,[5] and have the ability to form amyloids[6,7] or other cellular functions.[8] Certain compositional biases play significant roles in human neurodegenerative diseases.[9,10]

The great success of recent genome projects is an important factor in the development of algorithms and tools for the automated annotation of biological sequences. In recent years, in addition to a large number of prokaryotic genomes, the genomes of thousands of different eukaryotic species have been sequenced and assembled. The almost exponential growth of genomic and proteomic data is an important incentive for the development of algorithms and tools for the automated annotation of biological sequences. Some algorithms for the annotation of CB regions have previously been derived. The general aim of these existing algorithms has been the selective masking of CB regions without affecting other regions that could be potentially important. Examples of such algorithms are SEG (Segment Sequence(s) by Local Complexity) and CAST (Complexity Analysis of Sequence Tracts).[11,12] Later, a method for the identification of CB regions based on defining the lowest-probability subsequences (LPSs) for a given amino-acid composition was proposed.[6] This algorithm (referred to below as the original LPS algorithm) is based on scanning along the input sequence in a decreasing series of moving windows whose range of window sizes is specified by the user. The next adaptation of this method was an algorithm for the complete annotation of multiple amino acid residue biases.[8] This algorithm provides an exhaustive assignment of CB regions with a precise localization of boundaries. It was employed for the development of the LPS-annotate server and "Prion Home" database.[13,14] The LPS algorithm is a very useful and powerful tool for the annotation of LPSs, but some of its features reduce its efficiency. For examples, it uses an enumerative technique in which LPSs are found by checking all possible subsequences from 25 residues to the full sequence length with a step size of 1 residue.[6] In this algorithm, the dependence of processing time on the length of the sequence is nearly quadratic. Annotation of CB regions requires a lot of time and resources, especially for relatively long sequences.

Considering that processing time is essential for the processing of large datasets, we developed Sequence Analysis based on the Ranking of Probabilities (SARP): a novel algorithm for the annotation of LPSs. SARP provides a precise annotation of LPSs; it finds all of the LPSs that would be found by the original LPS algorithm. Our algorithm is based on ranking subsequences by their probabilities, followed by the selection of LPSs, which avoids enumeration. We achieved an approximately 230-fold faster performance with a dependence on sequence length that is closer to a linear relationship. SARP is especially useful for the processing of large datasets, such as sets of eukaryotic proteomes, as it permits the user to drastically reduce the computing time and hardware requirements for computation.

Methods

Materials

Algorithms were implemented using C#. All calculations were performed on a computer with a single 2.8 GHz Intel Core i7 CPU and 6 GB RAM. The probability threshold in all cases was 10^{-12}. The minimum window size for both algorithms was 25 aa, and the maximum window size for the original LPS algorithm was 1000 aa. The source code for SARP is available upon request.

All protein sequences were downloaded from the NCBI RefSeq database (http://www.ncbi.nlm.nih.gov/refseq/). To generate the sets of 1000 proteins of yeast *Saccharomyces cerevisiae*, we selected top 1000 proteins from the list of proteins sorted by accession. To generate the sets of 250 proteins for each of 5 species: *Homo sapiens*, *Drosophila melanogaster*, *Caenorhabditis elegans*, *Nanoarcheum equitans* and *Saccharomyces cerevisiae*, we sorted the list of the proteins by accession and selected the first protein of each n proteins. The parameter n was different for each species, so the sampling procedure covered the whole proteome.

Original LPS algorithm

We used the LPS algorithm described previously.[6] This algorithm is based on calculating the probabilities for all subsequences of the given sequence. To generate the set of subsequences, the authors used sliding windows of different sizes. For each individual amino acid of type *x*, the whole range of window sizes and for all positions of the window, the probability of subsequences was calculated as follows:

$$P(S_{i,w}, x) = \frac{w!}{n!(w-n)!} p_x{}^n (1-p_x)^{w-n} \qquad (1)$$

where $S_{i,w}$ is a subsequence of sequence S starting at position *i* with length *w, n* is the count of amino acid residues of type *x* in subsequence $S_{i,w}$ and px is the frequency of residues of this type in the whole proteome. The subsequence with the lowest probability was selected as the LPS, and the remainder of the sequence was resubmitted to the procedure. The input sequence was searched for LPSs for each residue with a preassigned range of window sizes (from 25aa to 1000aa).

Description of SARP

Unlike original LPS algorithm, our algorithm, SARP, finds short subsequences, which are likely to be the parts of LPSs, and extends them to cover the whole LPSs. A flow diagram of SARP is shown in Fig. 1. First, SARP finds the lower limit of LPS length for the given sequence. The procedure for the optimization is described below. Then, the window of the calculated length moves along the sequence with a step size of one residue, and both the count of x-type amino acids and the probability are calculated for each subsequence. Probability is calculated as for the original algorithm (1).

Next, all subsequences are split into groups so that all elements in each group have equal probabilities, and the groups are added to the queue in order of ascending probability. So, at the beginning of the queue, we have the group of subsequences with the lowest probability among the all subsequences of the same size, which are more likely to be the parts of LPSs. Then, the first group is taken from the queue and split into subgroups in such a manner that each subsequence in the subgroup overlaps with at least one other subsequence in the subgroup. We expect that all members of each subgroup are the parts of one LPS, if they are. Next, SARP determines the boundaries and probability of the LPSs covering the subgroups. For each subgroup, the first and last subsequences are selected, and each of these sequences is extended towards the beginning and end of the sequence as follows. At each step, one residue is added to the subsequence, and a new probability is calculated until the extension reaches the border of the sequence or the explored subsequence. Then, the subsequence with the lowest probability is chosen. Next, the extensions

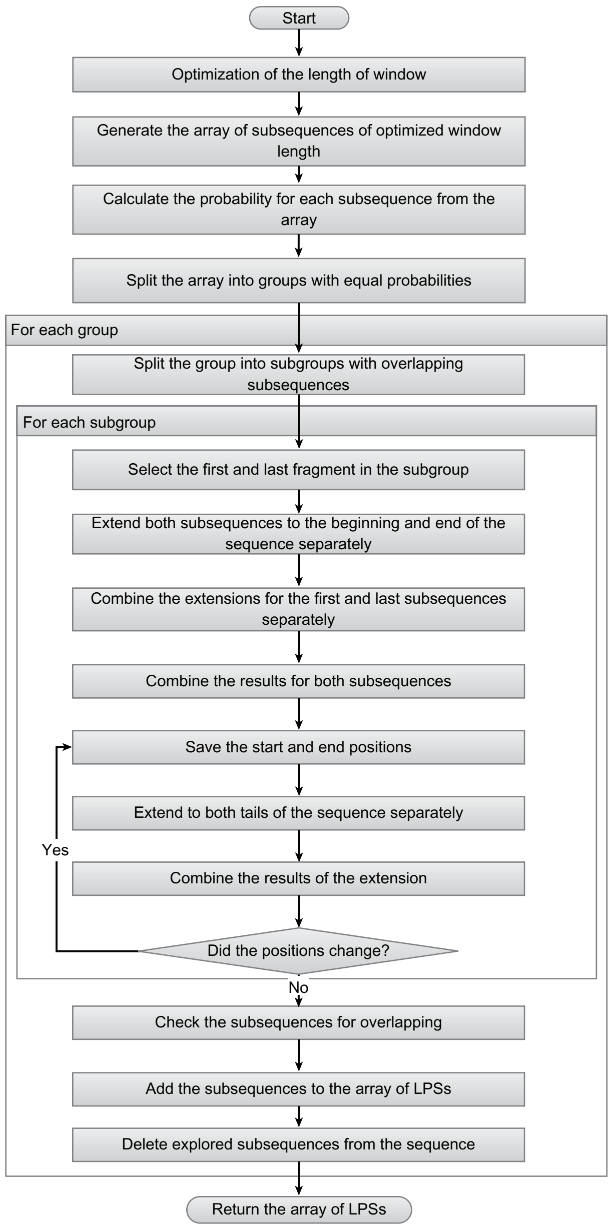
Start
Optimization of the length of window
Generate the array of subsequences of optimized window length
Calculate the probability for each subsequence from the array
Split the array into groups with equal probabilities
For each group
Split the group into subgroups with overlapping subsequences
For each subgroup
Select the first and last fragment in the subgroup
Extend both subsequences to the beginning and end of the sequence separately
Combine the extensions for the first and last subsequences separately
Combine the results for both subsequences
Save the start and end positions
Extend to both tails of the sequence separately
Yes
Combine the results of the extension
Did the positions change?
No
Check the subsequences for overlapping
Add the subsequences to the array of LPSs
Delete explored subsequences from the sequence
Return the array of LPSs

Figure 1. A workflow of SARP. The steps of SARP performance are shown below. The beginning and end of the algorithm are indicated. The steps suggesting alternative solutions are indicated with rhombs.

are combined as follows. We select the subsequence with the lowest probability from the extensions to the beginning, to the end and the union of both extensions. Then, from the first and last extended subsequences of the subgroup, the subsequence with the lowest probability is selected. For this subsequence, the procedure of extending to both ends and selecting the extension with the lowest probability is repeated until the beginning and end of the subsequence do not change. The LPS at each step is saved.

The LPSs generated for the subgroups, could overlap. To resolve overlapping, SARP uses the following procedure. For the group, the lowest probability subsequences from the last extension step for each subgroup with a probability lower than the threshold are checked for overlap. If there is a pair of overlapping subsequences, the subsequence with the higher probability is substituted with the LPS from the previous step for that subgroup, and checking is repeated. If the probability of the new LPS is higher than the threshold, it is excluded from the list of LPSs. The explored sequences for each group are excluded from further analysis. The final list of LPSs for each group is added to the final list of LPSs, which is returned by the algorithm. The example of running SARP is shown in Fig. 2.

Length optimization procedure

SARP outputs only the LPSs, whose probabilities are lower than the user-defined threshold, and whose lengths are bigger than the user-defined limit. We conclude that if the probability for every subsequence of the smallest window's length is much higher than the threshold, we can increase the length of the window until the probability of at least one of the subsequences is closer to the threshold. Therefore, we developed the window length optimization procedure. The length optimization procedure works as follows: SARP calculates the probability for an initial length w of each subsequence $S_{i,w}$ of sequence S (1). Then, if the lowest probability is greater than the square root of the threshold, SARP doubles the length; otherwise, it returns w. Given two subsequences of S, $S_{i,w}$ and $S_{i+w,w}$, that contain residues n_1 and n_2 of type x, the probability of subsequence $S_{i,2w}$ is:

$$p(S_{i,2w},x)=\frac{(2w)!}{(n_1+n_2)!(2w-n_1-n_2)!}p_x^{\,n_1+n_2}(1-p_x)^{2w-n_1-n_2} \tag{2}$$

Let $P(S_{i,w},x)>\sqrt{t}$ and $P(S_{i+w,w},x)>\sqrt{t}$, where t is the threshold. Then,

$$p(S_{i,w},x)P(S_{i+w,w},x)>t \tag{3}$$

and

$$\frac{P(S_{i,2w},x)}{P(S_{i,w},x)P(S_{i+w,w},x)}=\frac{(2w)!n_1!n_2!(w-n_1)!(w-n_2)!}{(w!)^2(n_1+n_2)!(2w-n_1-n_2)!}\geq 1 \tag{4}$$

Taking Equations (3) and (4) together, we can conclude that $P(S_{i,2w},x)\geq P(S_{i,w},x)P(S_{i+w,w},x)>t$. This proves the correctness of the optimization procedure.

Results

Faster algorithm with the same accuracy

The original LPS algorithm uses sliding windows, which have sizes that vary over a broad range, and a binomial distribution to calculate the probability of each subsequence.[6] The employment of sliding windows with sizes that vary from several residues to thousands requires the calculation of approximately *lr* probabilities, where *l* is the length of the protein sequence and *r* is the difference between the minimum and maximum sizes of sliding window. However, the maximum size of a LPS is equal to the length of the sequence; thus, the upper limit for the calculation of probability is near l^2. SARP does not use sliding windows of all sizes to find LPSs, which notably reduces computing time.

We benchmarked SARP with the original LPS algorithm using 1000 random protein sequences from *Saccharomyces cerevisiae*. The average length of a protein sequence in this set was 489.2 residues, and the total length was 489,217 residues. SARP was as accurate as the algorithm described by Harrison and Gerstein. It found 100% of the LPSs identified by the original algorithm, and the boundaries of those LPSs were principally the same as for the original algorithm (Fig. 3, Supplementary File S1). The exception was cases of LPSs that were longer than the maximum window size. In contrast with the original LPS algorithm, SARP does not have an upper limit of window size, enabling the prediction of longer LPSs than the original algorithm can handle. We found only 1

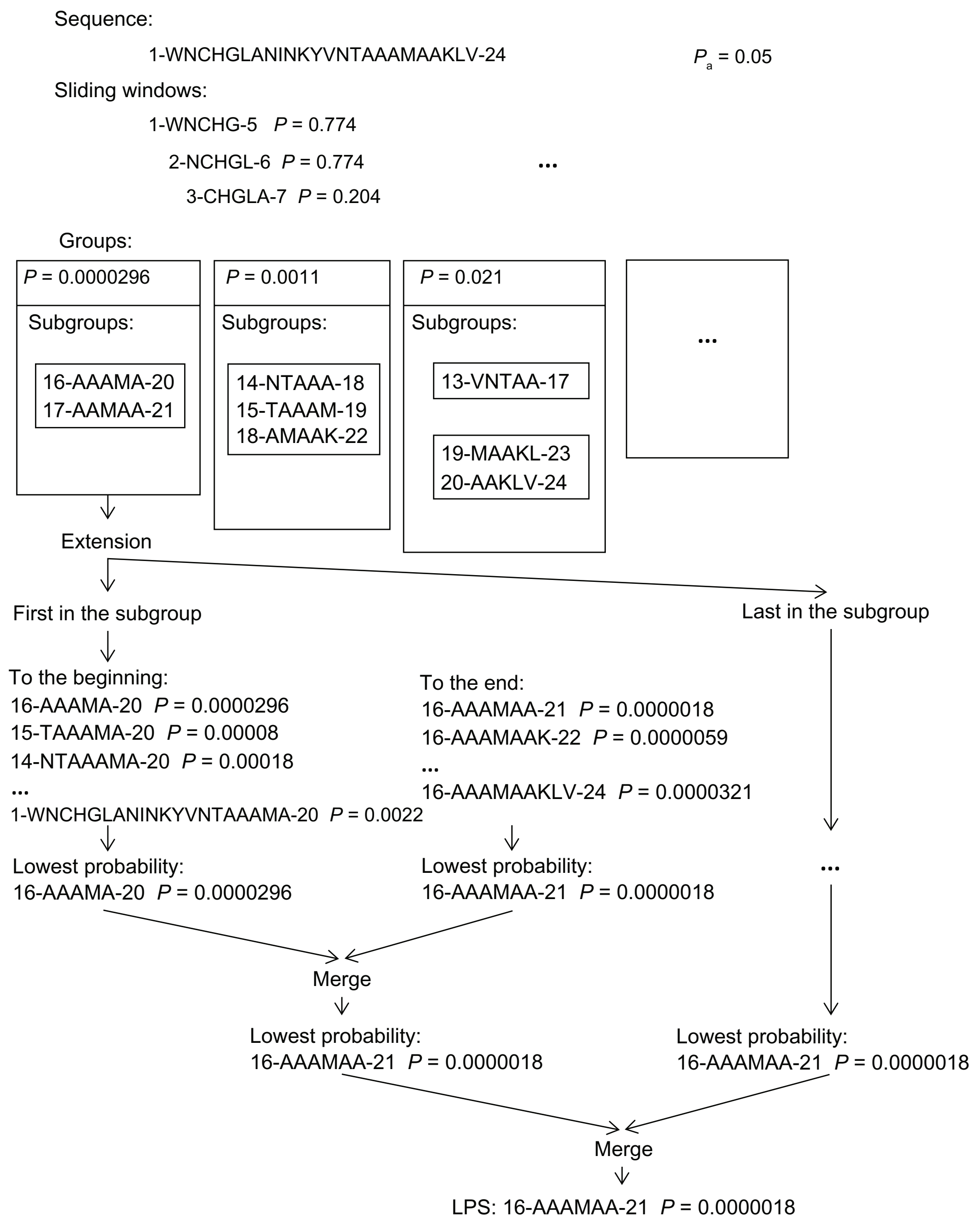

Figure 2. The scheme of search for LPS in example sequence with SARP for the amino acid alanine (A). The groups of the fragments with equal probability and the subgroups are shown. The process of extension is demonstrated for the first group.

that was inaccurate but it was very close to the original prediction of LPS boundaries for a very long LPS with an amino acid frequency that was lower than average for protein with GI 6319255.

We compared computation times between the original LPS algorithm and SARP. As expected, SARP demonstrated dramatic reductions in computation time. The tested set of 1000 proteins was processed in 1465 seconds of running time, ie, approximately half an hour, whereas for the original LPS algorithm, this index was 341,914 seconds (approximately 4 days). Thus, SARP is approximately 230-fold faster than the original algorithm. A summary of the time consumed by each algorithm to find the LPSs in 1000 yeast proteins is presented in Table 1.

Computation time strongly depends on protein length

It is clear that the computation time required to find the LPSs within a protein sequence depends on the length of the sequence. Over the range of protein lengths in our testing set (Fig. 4C), the

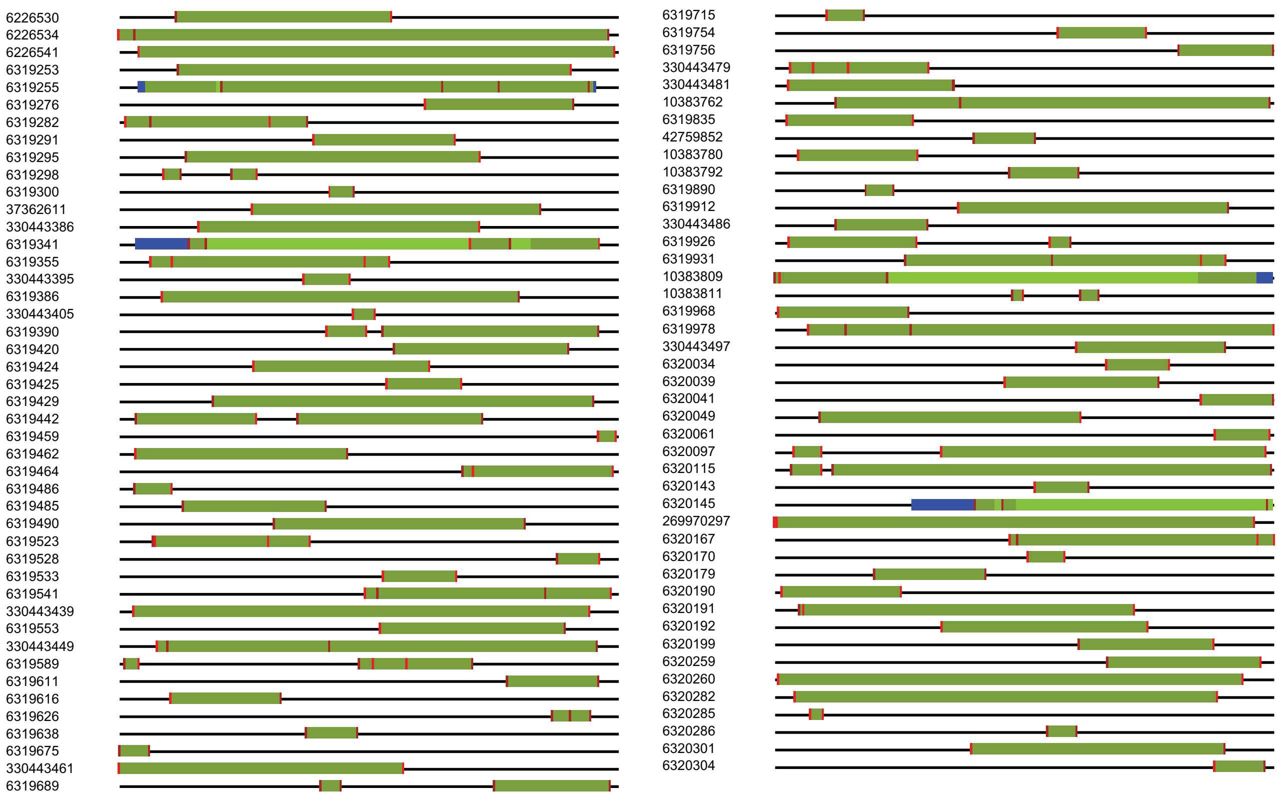

Figure 3. The scheme represents all LPSs found with the original LPS algorithm and with SARP in a set of 1000 yeast proteins. The numbers are the GI numbers of proteins in the NCBI database. The horizontal black line represents a protein sequence. Different green regions represent overlapping LPSs found with the original algorithm and SARP. Blue regions denote parts of an LPS that were not identified by the original algorithm. Vertical red dashes denote an exact match of the LPS boundaries found with the original LPS algorithm and SARP.

computation time of the original algorithm increases very quickly, reaching up to approximately 2 hours for a sequence of 2800 residues (Fig. 4B), whereas SARP required only approximately 12 seconds for the same sequence length (Fig. 4A). To better understand this dependence, we sorted all sequences into several groups by length with an increment of 200 residues (Fig. 4C). A comparison between SARP and the original LPS algorithm in processing times for proteins grouped by size is shown in Figure 4D. Because the difference in computation time for SARP and the original LPS algorithm is too large to illustrate both in the same histogram, an additional histogram for SARP alone is included (Fig. 4E). As discussed above, the upper limit of runtime of the original algorithm is close to l^2. Comparing the running times for SARP and the original LPS algorithm for groups of different protein lengths (Fig. 4F), we have shown that the ratio between the running times for the original LPS algorithm and SARP depends on the length of protein linearly. Thus, the dependence between the SARP runtime and sequence length is closer to linear than for original LPS algorithm. Taken together, SARP provides significantly faster performance, and its advantage is most prominent for long sequences.

Another question was whether the computing time for SARP depends on the frequencies of amino acids within the proteome. To test this, we compared the speed of LPS detection between proteins from 5 different species. We selected 250 random proteins

Table 1. A comparison of the efficiency between SARP and the original LPS algorithm.

Parameter	LPS algorithm	SARP
Total protein length, aa	489217	489217
Total time, ms	341914177	1464619
Average protein length, aa	489.2	489.2
Average time per protein, ms	341914.2	1464.619
Times faster*	1	233.45

Note: *This parameter illustrates the ratio of running times between the LPS algorithm and SARP, in which the running time of the LPS algorithm is set to 1.

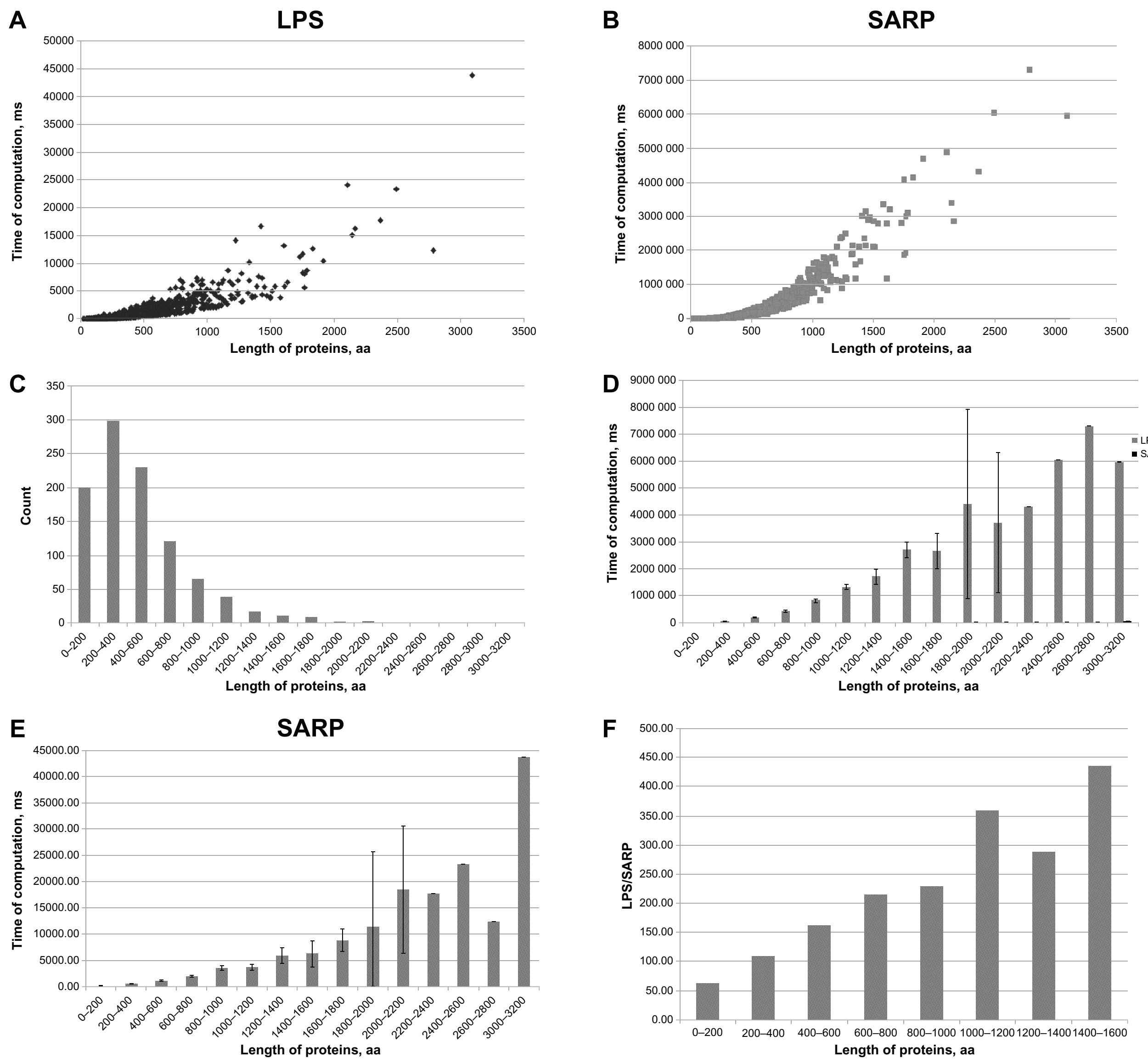

Figure 4. (**A**) A distribution of computation times for separate proteins of different lengths using SARP. Lengths of protein sequences (aa) and times of computation (ms) are shown. (**B**) The same as A. for the original LPS algorithm. (**C**) A histogram of relative numbers of proteins from the set of 1000 sequences that were analyzed grouped by their lengths. (**D**) A comparison of CPU running times for the original LPS algorithm and SARP dependent on the length of proteins. The columns of SARP results are nearly invisible due to its very fast computation time relative to the original algorithm. The results are indicated as the mean ± the confidence interval ($P \geq 0.95$). (**E**) Special histogram for SARP computation times. The results are indicated as the mean ± confidence interval ($P \geq 0.95$). (**F**) Ratio between the CPU times for the original LPS algorithm and SARP in the groups of proteins arranged by their length (aa).

from *Homo sapiens*, *Drosophila melanogaster*, *Caenorhabditis elegans*, *Nanoarcheum equitans* and *Saccharomyces cerevisiae*. Those species are from different taxonomic groups, live in different habitats and possess different compositions and average protein lengths (Fig. 5). The average protein lengths for the selected organisms vary from 285 residues for *N. equitans* to 661 for *D. melanogaster* (Fig. 5B), with a large spread of amino acid frequencies between the 5 sets of proteins tested (Fig. 5A). We compared the running times for the original algorithm and SARP separately for each of the 5 sets of proteins. If the average protein length is the major factor defining the performance of the algorithm, the approximate ratio of running times should be the same as the ratio of average sizes. If the running time depends mostly on the distribution of amino acids, we may expect a violation of this correlation. Our analyses demonstrated that for both SARP and the original LPS algorithm, the CPU running time strongly correlates with the average protein length and does not depend on the composition of amino acids (Fig. 5C and D).

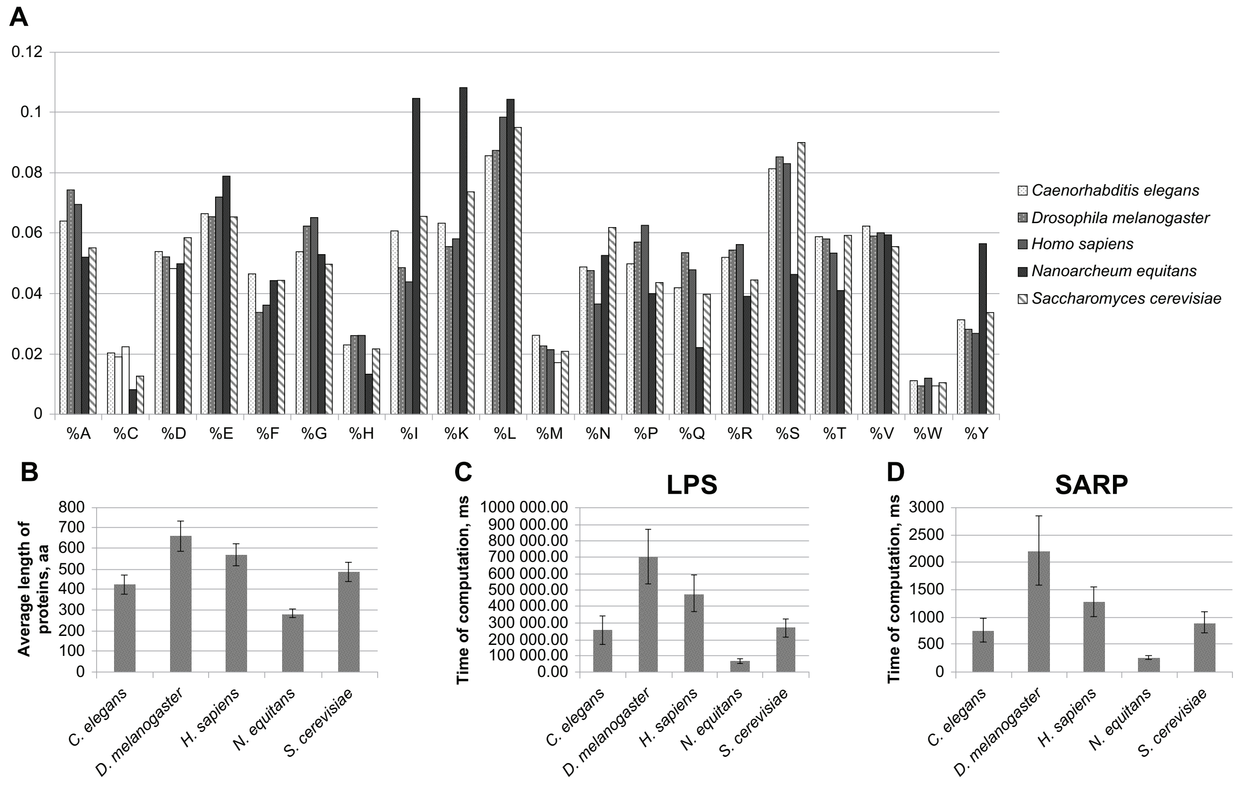

Figure 5. (**A**) The frequencies of amino acids for the sets of 250 proteins for each of the five species analyzed. The means of frequencies are indicated as percentages. (**B**) The average protein length (aa) is indicated for the set of sequences from each species. (**C**) Computation time using the original LPS algorithm for the sets of 250 proteins from five different species. Computation time is indicated in ms. The results are shown as the mean ± the confidence interval ($P \geq 0.95$). (**D**) The same as C. for SARP.

The runtimes for the subsets containing proteins of the same length from different organisms were very close to each other (data not shown). Taken together, these data confirm the efficiency of SARP for the processing of large datasets.

Discussion

The role of amino acid composition was first discussed many years ago.[15] Since then, shifts in the composition of protein sequences have been attributed to a large number of interesting phenomena of the living world, including adaptations to extreme ecological niches by Archaea and bacteria,[1,2] prions and amyloids[6,16] and many others. Analysis of composition is used to characterize the structure,[13] functions and evolution of proteins.[8,17] There is evidence suggesting that the composition of amino acids is a unique "molecular signature" of each species, similar to the GC content of genomes.[1] However, unlike GC content, the amino acid composition of proteomes has been studied very poorly to date, primarily due to the relatively weak elaboration of algorithms for systemic annotation of compositional biases in proteins. It is noteworthy that there are a lot of papers describing shifts in the composition of entire proteomes or individual proteins, but that the number of papers dedicated to the systematic analysis of CBs in proteins is much lower. One of the most convenient algorithms allowing such analysis is the LPS algorithm.[6] It was successfully utilized in the original paper for the annotation of potentially amyloidogenic and prionogenic proteins based on their enrichment in asparagine and glutamine, which is the common feature of prions. Furthermore, this algorithm was used for the "Prion Home" (http://libaio.biol.mcgill.ca/prion)[14] and "LPS annotate" (http://cedra.biol.mcgill.ca/lps-annotate.html) databases[13] and for a series of studies dedicated to the characterization of CB regions. These implications of the LPS algorithm confirm its relevance. Our algorithm, SARP, can also be used in such studies with the advantage of very fast processing times. Our algorithm permits the annotation of LPSs in eukaryotic proteomes within several hours of CPU running time on a single CPU machine without using high-performance computer systems. SARP may be useful for any web resources demanding permanent annotation of novel LPSs, such as “LPS annotate”, because compared with the LPS algorithm, it strongly reduces the required computation capacity.

To conclude, we herein described the use of SARP, a novel algorithm for assessing compositional biases. This algorithm was shown to have high fidelity, allowing the precise identification of CB regions in proteomes. Indeed, comparing the original LPS algorithm and SARP for 1000 yeast proteins, we found only 1 protein for which the original algorithm was more precise than SARP. We did not filter the LPSs by the frequency of residues of type *x* within them. So, the output of the both algorithms contained LPSs of 2 kinds, with increased and with decreased abundance of residues of type *x*. The inaccurate LPSs found by SARP belong to the second class. All LPSs with increased abundance of the *x*-type residues were found accurately by SARP. So, this inaccuracy can be easily overcome by searching not for LPSs with decreased abundance of the *x*-type residues, but for LPSs with increased abundance of non-*x*-type residues.

The general advantage of this algorithm is its significantly faster performance. The basis of such a performance improvement is the method of identification of LPSs. In contrast with the previously published LPS algorithm, SARP does not use enumeration of subsequences but directly ranks their probabilities and uses a new procedure of optimization of window lengths. A performance test showed that CPU running time with SARP was approximately 230-fold faster than the original LPS algorithm. For the set of 1000 yeast’s proteins we achieved the reduction of the runtime up to approximately 95 hours, using SARP instead of the original algorithm. We could expect that the typically used data sets are the whole proteomes, which are much larger. The whole proteome of *S. cerevisiae* is about 6000 proteins, and the proteomes of mammals are even bigger, up to approximately 35000 proteins for human proteome. Therefore, the gain of runtime would be even bigger for normal data sets.

Thus, SARP is a powerful tool for the annotation of LPSs in large datasets, which is important for comparative, functional and evolutional proteomics.

Author Contributions

Conceived and designed the experiments: KSA, AAN. Designed the algorithm: KSA. Analyzed the data: KSA, AAN. Wrote the first draft of the manuscript: AAN. Contributed to the writing of the manuscript: KSA. Agree with manuscript results and conclusions: KSA, AAN. Jointly developed the structure and arguments for the paper: AAN. Made critical revisions and approved final version: KSA, AAN. All authors reviewed and approved of the final manuscript.

Funding

The study was supported by the Ministry of Education and Science of Russia, project 14.132.21.1324. This work was supported by a grant from the St. Petersburg Government.

Competing Interests

Author(s) disclose no potential conflicts of interest.

Disclosures and Ethics

As a requirement of publication the authors have provided signed confirmation of their compliance with ethical and legal obligations including but not limited to compliance with ICMJE authorship and competing interests guidelines, that the article is neither under consideration for publication nor published elsewhere, of their compliance with legal and ethical guidelines concerning human and animal research participants (if applicable), and that permission has been obtained for reproduction of any copyrighted material. This article was subject to blind, independent, expert peer review. The reviewers reported no competing interests.

References

1. Schmidt A, Rzanny M, Schmidt A, Hagen M, Schütze E, Kothe E. GC content-independent amino acid patterns in bacteria and archaea. *J Basic Microbiol*. 2012;52(2):195–205.
2. Tadeo X, López-Méndez B, Trigueros T, Laín A, Castaño D, Millet O. Structural basis for the aminoacid composition of proteins from halophilic archea. *PLoS Biol*. 2009;7(12):e1000257.
3. Long JC, Caceres JF. The SR protein family of splicing factors: master regulators of gene expression. *Biochem J*. 2009;417(1):15–27.
4. Neduva V, Russell RB. Proline-rich regions in transcriptional complexes: heading in many directions. *Sci STKE*. 2007;2007(369):pe1.
5. Uversky VN, Dunker AK. Understanding protein non-folding. *Biochim Biophys Acta*. 2010;1804(6):1231–64.
6. Harrison PM, Gerstein M. A method to assess compositional bias in biological sequences and its application to prion-like glutamine/asparagine-rich domains in eukaryotic proteomes. *Genome Biol*. 2003;4(6):R40.

7. lberti S, Halfmann R, King O, Kapila A, Lindquist S. A systematic survey identifies prions and illuminates sequence features of prionogenic proteins. *Cell*. 2009;137(1):146–58.
8. Harrison PM. Exhaustive assignment of compositional bias reveals universally prevalent biased regions: analysis of functional associations in human and Drosophila. *BMC Bioinformatics*. 2006;7:441.
9. Gitler AD, Shorter J. RNA-binding proteins with prion-like domains in ALS and FTLD-U. *Prion*. 2011;5(3):179–87.
10. Orr HT. Polyglutamine neurodegeneration: expanded glutamines enhance native functions. *Curr Opin Genet Dev*. 2012;22(3):251–5.
11. Wootton JC, Federhen S. Analysis of compositionally biased regions in sequence databases. *Meth Enzymol*. 1996;266:554–71.
12. Promponas VJ, Enright AJ, Tsoka S, et al. CAST: an iterative algorithm for the complexity analysis of sequence tracts. Complexity analysis of sequence tracts. *Bioinformatics*. 2000;16(10):915–22.
13. Harbi D, Kumar M, Harrison PM. LPS-annotate: complete annotation of compositionally biased regions in the protein knowledgebase. *Database (Oxford)*. 2011;2011:baq031.
14. Harbi D, Parthiban M, Gendoo DM, et al. PrionHome: a database of prions and other sequences relevant to prion phenomena. *PLoS ONE*. 2012;7(2): e31785.
15. Conn EJ, Berggren RE. Studies in the physical chemistry of proteins: III. The relation between the amino acid composition of casein and its capacity to combine with base. *J Gen Physiol.* 1924;7:45–79.
16. Michelitsch MD, Weissman JS. A census of glutamine/asparagine-rich regions: implications for their conserved function and the prediction of novel prions. *Proc Natl Acad Sci U S A*. 2000;97(22):11910–5.
17. Harrison LB, Yu Z, Stajich JE, Dietrich FS, Harrison PM. Evolution of budding yeast prion-determinant sequences across diverse fungi. *J Mol Biol*. 2007;368(1):273–82.

Supplementary material

Supplementary File 1
List of LPSs found by SARP and by the original LPS algorithm.