An algorithm to find all identical internal sequence repeats

Nirjhar Banerjee[1], N. Chidambarathanu[1], Daliah Michael[1], N. Balakrishnan[2] and K. Sekar[1,2,*]

[1]Bioinformatics Centre (Centre of Excellence in Structural Biology and Bio-computing) and
[2]Supercomputer Education and Research Centre, Indian Institute of Science, Bangalore 560 012, India

Proteins containing amino acid repeats are considered to be of great importance in evolutionary studies. The principal mechanism of formation of amino acid repeats is by the duplication or recombination of genes. Thus, repeats are found in both nucleotide and protein sequences. In proteins, repeats are involved in protein–protein interactions as well as in binding to other ligands such as DNA and RNA. The study of internal sequence repeats would be helpful to scientists in various fields, including structural biology, enzymology, phylogenetics, genomics and proteomics. Hence an algorithm (Finding All Internal Repeats, FAIR) has been designed utilizing the concepts of dynamic programing to identify the repeats. The proposed algorithm is a faster and more efficient method to detect internal sequence repeats in both protein and nucleotide sequences, than those found in the literature. The algorithm has been implemented in C^{++} and a web-based computing engine, IdentSeek, has been developed to make FAIR accessible to the scientific community. IdentSeek produces a clear, detailed result (including the location of the repeat in the sequence and its length), which can be accessed through the world wide web at the URL http://bioserver1.physics.iisc.ernet.in/ident/

Keywords: Dynamic programing, evolutionary studies, internal sequence repeats, structure–function relationship.

INTERNAL repeats occur in approximately 14% of all known proteins available in the literature. In fact, eukaryotic proteins are three times more likely to contain internal sequence repeats than prokaryotic proteins, because eukaryotic repeats have unique functions[1]. In addition, eukaryotic genomes have undergone a higher degree of gene duplication. Since gene duplication and recombination events are thought to be responsible for the existence of sequence repeats, they are found in both genes and non-coding genomic regions. In protein sequences, repeats can range from a single residue to complete domains consisting of hundreds of amino acid residues. Repeats that are separated by intermediate sequences of constant length occurring in clusters are referred to as short regularly spaced repeats (SRSRs)[2]. Since most of these are identical repeats, they can be effectively applied to protein secondary-structure prediction.

Earlier studies indicated that the topology of transmembrane proteins[3] evolved by internal gene duplication[4,5] and a large number of them have been detected to contain internal sequence repeats[5–7]. As stated earlier, the secondary structures of such internal repeats in proteins are regular[8]. A classic example of internal repeats is the zinc-finger domain, which is a commonly repeated motif in most DNA-binding proteins. Internal sequence repeats also have important implications in evolution and functions of proteins. In his book, *Of Flies, Mice and Men*, Francois Jacob has put forth that evolution has arisen from nature's tinkering, implying that nature makes new genes with new functions from existing genes. Kachroo *et al.*[9] suggested that the formation of internal sequence repeats could be the major cause of evolutionary genome variability. Furthermore, in evolutionary studies, the distance of the constituent repeats can use information from internal repeats to estimate the age of a genome or an individual protein[10]. In addition, insights into evolution and phylogenetic relationships can be derived from the distribution of repeats. For instance, Koonin *et al.*[11] concluded that the distribution of repeats in archaea indicates that they have an intermediate relationship between prokaryotes and eukaryotes. Earlier studies indicate that repeats play a major role in protein and nucleotide stability, thus being responsible for protein function[10]. Internal repeats not only ensure the proper functioning of proteins, but sometimes cause malfunction and hence are important in the study of diseases[10,12], such as Alzheimer's disease, Creutzfeldt–Jacob disease and Gerstmann-Straussler–Scheinker disease[13,14].

In the case of DNA sequences, short repeats indicate the position of deletion and precise removal of transposable elements[15]. Longer repeats are responsible for class switching in immunoglobulins[16]. Further, WD40 repeats in CARD4 act as regulators for the activation of caspases in apoptosis pathways[17]. The activity of enzymes can also be controlled to some extent by internal sequence repeats. For example, decay accelerating activity is reduced if repeats between sites 1 and 2 are deleted in C5 converta-

*For correspondence. (e-mail: sekar@physics.iisc.ernet.in)

ses[18]. Further, in the case of cGMP-binding phosphodiesterases there are two repeats, each of which is a catalytic site and an invariant Aspartate residue is essential for interaction[19]. Furthermore, in case of nucleotide repeats, telomeric repeats are essential to ensure correct function of telomeres and for maintaining the integrity and stability of the chromosome[20]. Thus, the significance of internal sequence repeats is clear.

Existing algorithms

Many algorithms have been developed to search for sequence repeats in protein sequences. These methods use different methodologies and protocols to detect internal sequence repeats. According to Söding *et al.*[21], there are three general classes among the existing methods to detect repeats in protein sequences. SMART[22] and REP[23] belong to the first class, which construct repeats by utilizing their own database of profile-hidden Markov models or sequence profiles generated from known repeat families and comparing each profile with the query sequence. REPPER[24] belongs to the second class which is specialized for the detection of periodic patterns in proteins. This class does not allow gaps within or between repeats and is applicable mostly to the large class of fibrous proteins. Although REPwin in REPPER is similar to other tools, it detects short repeats by aligning the sequence against itself and deploying a sliding window protocol. However, these short repeats are detected only if they occur consecutively in sufficient number in the sequence to be detected by Fourier transformation. The third class consists of REPRO[25], RADAR[26] and TRUST[27]. These do not rely on any a priori knowledge about repeat families, but look for internal similarities by aligning the sequence with itself. RADAR results in short composition-biased repeats together with gapped approximate repeats. TRUST generates only those repeats that are statistically significant. The program REPuter[28] finds repeats only in nucleotide sequences by implementing suffix trees.

Most of these methods concentrate on homology and produce repeats more than 15 residues that frequently contain gaps. Thus, small repeats may be missed by some of these programs. While some programs allow mismatches to account for the mutations in sequences, and are a good design choice for the detection of domain repeats across different proteins, they have the disadvantage of sometimes showing completely unrelated sequences as repeats. Additionally, the main limitation of these algorithms is the limit imposed on the maximum number of amino acid residues that can be given as input. RADAR web-interface permits a maximum of 1000 amino acid residues, Internal Repeats Finder[29] permits 2000 amino acid residues and REPRO allows a maximum of 6000 amino acid residues. However, REPPER has no limits on the number of amino acid residues in a given sequence, but permits only one protein sequence at a time, completely ignoring any later sequences provided in the input. Further, in case of RADAR, TRUST and REPRO, when multiple protein sequences are entered, the algorithm treats the entire set of sequences as a single sequence.

To address these issues, an algorithm FAIR (Finding All Internal Repeats) has been proposed to search for all identical internal sequence repeats. In contrast to the algorithms described earlier, the proposed algorithm utilizes the concepts of dynamic programing rather than simple sequence alignment to find the internal identical repeats in both protein and nucleotide sequences. Unlike BLAST, which is a local alignment search tool and whose output must be heavily parsed to find internal repeats in sequences, the proposed algorithm finds and individually lists all the internal repeats present in the sequence. Further, the algorithm runs well for finding precise results from extremely large sequences, as described in later sections.

The proposed algorithm

The algorithm to find identical internal sequence repeats in protein and nucleotide sequences is described below. The time taken for execution and the memory utilized while running the program have been optimized and are explained in the following sections. As the number of strings repeated and the number of times one particular string is repeated are not known, size allocation of arrays is not preferred, and thus the simple dynamic programing concepts are utilized in this algorithm. The algorithm consists of three parts and is explained in detail in the subsequent sections.

Finding the repeats using two vectors, 'current' and 'previous'

The uploaded sequence or string of length y is named $a1$ and an identical string $a2$ is created for calculation purposes. Two vectors named 'previous' and 'current' of $y + 1$ elements each are created and initialized to zero. The $[y + 1]$th element in the vector is required in order to check for the end of the string. Although there is no alignment involved, the procedure to find identical internal repeats in a given sequence can be visualized using a graph, where the string $a1$ is along the y-axis and $a2$ is along the x-axis (Figure 1). For each character of $a1$, the algorithm checks all the characters of $a2$. In effect, a match between the character denoting the row (in string $a2$) and the one denoting the column (in string $a1$) is searched. Since the matrix is symmetrical, it is sufficient to check only the upper half of the main diagonal. In other words, if i denotes the position of the pointer in $a1$, the algorithm checks from the $[i + 1]$th element of $a2$ till the end of the corresponding row and whenever a match is found, the following operation is performed:

if ($a1[i]$ equals $a2[j]$){
set current[j] to previous[$j-1$] + 1;}

The above step assigns the current length of the repeat to the jth element of vector 'current'. While performing the next iteration, the vector 'previous' is assigned the value of vector 'current' and the above step is repeated. For example, if the string 'ABC' (position 6–8) is repeated twice, with the second repeat from positions 11 to 13 in the vector, the changes in the required elements of both vectors will be as shown below:

1. Initially, current = 0; previous = 0.
2. After the first match A: current[10] = 1; previous=0.
3. Then, previous is assigned the value of current: previous[10] = 1; current[10] = 1; rest = 0.
4. After the character B is found: current[11] = 2; no other changes.
5. Similarly, after the character C is found: current[12] = 3.

Thus, the 'current' vector stores both the position and length of the repeat. In this case, we find that 3 is the length of the repeat and the 13th position is the 'end point'. The algorithm first checks whether the last length of the repeat (if there is one) that is stored in the 'previous' vector, is greater than the minimum length (min1) of the repeat defined by the user. If this condition is satisfied, the position and length of the repeat are pushed into another vector, 'vsparse'. Subsequently, the vector 'vsparse' stores the user-defined data-type structure called 'SparseEntry' which in turn stores three variables, corresponding to the position, end (row_value and column_value) and length (length_value) of the repeat. Thus, the algorithm performs the following operation whenever a match is not found and if the end of a row is not reached:

if(previous[$j-1$] is greater than or equal to min1){
set sparse_entry_column_value to $j-1$;
set sparse_entry_row_value to $i-1$;
set sparse_entry_length_value to previous[$j-1$];
push sparce_entry into vector vsparse;
}

Once the end of the row is reached and the condition previous[j-1]>=min1 is satisfied, the algorithm stores the values in the vector as mentioned earlier. Thus, the values of the 'end-points' and the length of the repeats have been computed and stored at the end of the first part of the algorithm. When more than one sequence is given as input, the repeat is stored and the vector current is reset to 0. Further, whenever a new sequence is encountered, the vector 'previous' is set to the value of 'current'; hence it is also reset to zero.

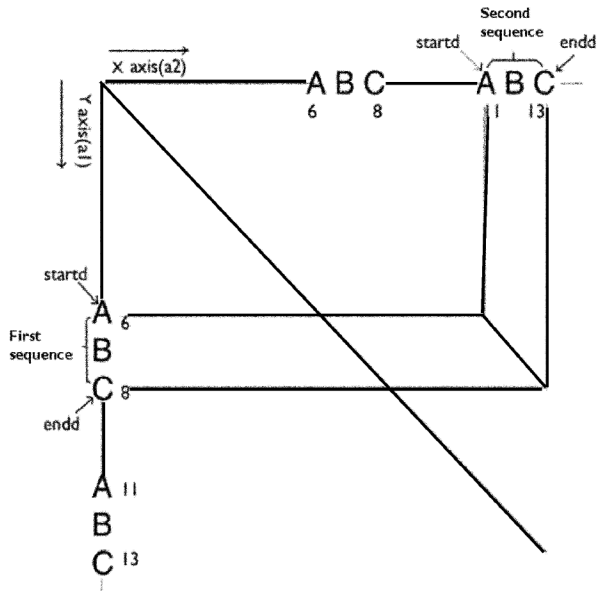

Figure 1. Visualization of the procedure to detect internal repeats (*ABC*) from a sequence. $a1$ and $a2$ are the two vectors containing the substring *ABC*, which is repeated between positions 6 and 8 as well as 11 and 13. 'Startd' and 'endd' contain the starting and ending positions of the substring respectively.

Storing subsequences and repeat positions

When the length of the repeat (stored in vsparse.value) is greater than the minimum length, the vectors 'startd' and 'endd' are created based on the contents of vsparse. This can be explained using the string '*ABC*', which is repeated between positions 6 and 8 as well as 11 and 13 (Figure 1). Thus, the vector 'startd' contains the positions of the starting point of the 'first' and the 'second' sequences. Similarly, the vector 'endd' contains the positions of the end-points of the two sequences. If there are more than two occurrences of a particular string, the 'startd' and 'endd' values of the 'second sequence' for further occurrences need to be stored, but not those of the 'first sequence'. To accomplish this, the algorithm uses a Boolean variable append_first, which is false unless it encounters a new repeat.

if ((endd of first sequence is not equal to lrow) OR
(seqlen is not equal to vsparse[i].value)) {
set append_first to true;
set seqlen to vsparse[i].value;
set lrow to endd[firstseq];
}

if(append_first is true){
set sub_sequence.start_point to startd[firstseq];
set sub_sequence.end_point to endd[firstseq];
push sub_sequence into vsubseq;
}

```
set sub_sequence.start_point to startd[secondseq];
set sub_sequence.end_point to endd[secondseq];
push sub_sequence into vsubseq;
```

Whenever either the end-points of two consecutive repeats or their lengths are different, append_first is set to true. A new vector 'vsubseq' stores a data structure called 'SubSequence' containing three elements: sequence (to store the repeat) and start_point and end_point (for the initial and terminal points of the repeat respectively). Thus, when a new repeat is encountered, the positions of both the 'first' and 'second' sequences are pushed into the vector 'vsubseq'. In other cases, only the positions of the 'second' sequence are pushed.

Sorting the vector and removing identical entries

The entries of the vector 'vsubseq' are entered as and when the repeats are encountered. Thus, the results must be sorted to make the output non-redundant. This is performed using the built-in-function STL sort (NlogN), implemented in C^{++}. Further, in order to remove redundant entries from the sorted output every element is checked with the previous one and if two subsequent entries are found to be the same, the second one is removed. The non-redundant entry is pushed into a vector called 'unique_entry', which displayed as the output. Interestingly, none of the generated sets of repeats is a subset of another. To illustrate this aspect, suppose the sequence '*ABCDE*' is repeated three times, it would also mean that the sequence '*ABCD*' is repeated three times. However, this second repeat ('*ABCD*') will not be shown unless there is an independent repeat of '*ABCD*'. Thus, the algorithm is designed to show only non-redundant repeats. However, in the case of low-complexity repeats the algorithm generates all subsets as repeats, irrespective of whether they occur independently in other parts of the sequence.

While FAIR follows $O(N^2)$ time complexity in the generalized case, where N is the number of amino acids present in the sequence, TRUST follows $O(N^2 + NA^2 + TLN)$ time complexity, which takes less than a minute to find repeats in 2000 residues[27]. While the algorithm used in RADAR takes not more than 180 s to find repeats in 5179 amino acid residues (e.g. MUC2_HUMAN) and 72 s to identify repeats in a sequence of length 2265 residues (e.g. FINC_BOVIN)[26], the proposed algorithm takes less than a second to find repeats from these sequences. Further, FAIR takes only 32 s to find repeats in the largest protein sequence 'Titin' from *Homo sapiens* (gi|17066105|emb|CAD12456.1), which contains 34,350 amino acid residues. Thus, the time taken by the proposed algorithm is less compared to those available in the literature. However, keeping in mind the time complexity, the algorithm (and web-based computing engine) performs best with a minimum repeat length of three amino acids.

Implementation: IdentSeek, a web-based computing engine

In order to make the proposed algorithm freely available to the scientific community, a web-based computing engine, IdentSeek, has been developed. The computing engine is written in Perl/CGI and can handle extremely large sequences (FASTA format). It has been successfully tested with the nucleotide sequence of the parallel β-helix repeat containing protein from *Chlorobium chlorochromatii* CaD3 (gi|78188592; approximately 37,000 amino acids; 110,000 nucleotides). In addition, it is possible to distinguish between different protein sequences when multiple such sequences are uploaded and to detect all identical repeats irrespective of restrictions on their composition or alignment. Further, it is flexible and allows the user to set the required length of the repeat. Finally, it produces a complete and comprehensive output for each of the sequences.

Case studies

To test the efficiency of the proposed algorithm, an invasion from *Yersinia pestis* Nepal516 containing 4270 amino acid residues was taken in FASTA format. The minimum number of amino acid residues in a given repeat was set to greater than or equal to 60. For clarity, only a portion of the input protein sequence is shown in Figure 2. It is evident from the output that there are two distinct

```
>gi|108813752|ref|YP_649519.1| invasin [Yersinia pestis Nepal516]
MLNYFRAILISWKWKLSHHTSRPHDVKEKGHPRKIKVVAWITLFFQFAFPLSLSFTPAIAAANTTNSAPT
SVITPVNASILPPAARATEPYTLGPGDSIQSIAKKYNITVDELKKLNAYRTFSKPFASLTTGDEIEVPRK
.....
.....

Number of residues in the repeat = 64

DGIATATLTNTVAGTSNVVATIDTVNANIDTAFVAGAVATITLTAPV
NGAVADGADTNQVDALV
[ 1002 to 1065 ]
[ 2452 to 2515 ]
--------------------------------------------------------------------------

Number of residues in the repeat = 248

DGIATATLTNTVAGTSNVVATIDTVNANIDTAFVAGAVATITLTAPV
NGAVADGADTNQVDALVEDANGNPITGAAVVFSSANGATILSSTMNT
GVNGVASTLLTHTVAGTSNVVATVDTVNANIDTTFVAGAVATITLTT
PVNGAVADGANSNSVQAVVSDSDGNPVTGAAVVFSSANATAQITTVI
GTTGADGIATATLTNTVAGTSNVVATIDTVNANIDTAFVAGAVATIT
LTAPVNGAVADGA
[ 1002 to 1249 ]
[ 2259 to 2506 ]
--------------------------------------------------------------------------
```

Figure 2. Output of FAIR for protein sequence using an invasion from *Yersinia pestis*.

Sequence	Start	Size
LGAGSQAHGSQSLALGAGATASQANSIALGASSVTTVGAESDYSAYGLTAPQTSVGEVGMGTAQG NRKITGVAAGSADYDVVNVAQLTAVGDKVEQNTADITSLGGRVTNVEGGMTRITNGGGIKYFHTH STEPDSVASGSDSVAIGPNAQASGTTSIA	1	159
MGAGSTAQGAQSLALGAGAAASQANSIALGASSVTTVGAESDYSAYGLTAPQTSVGEVGVGTAQG NRKITGVAAGSADYDAVNVAQLTAVGDKVDQNTADITSLDGRVTNVEGEMASITNGGGVKYFHTH STESDSVASGSDSVAIGPNAQASGTASVA	160	159

Figure 3. Output of TRUST server for part of the hypothetical protein from *Y. pestis*.

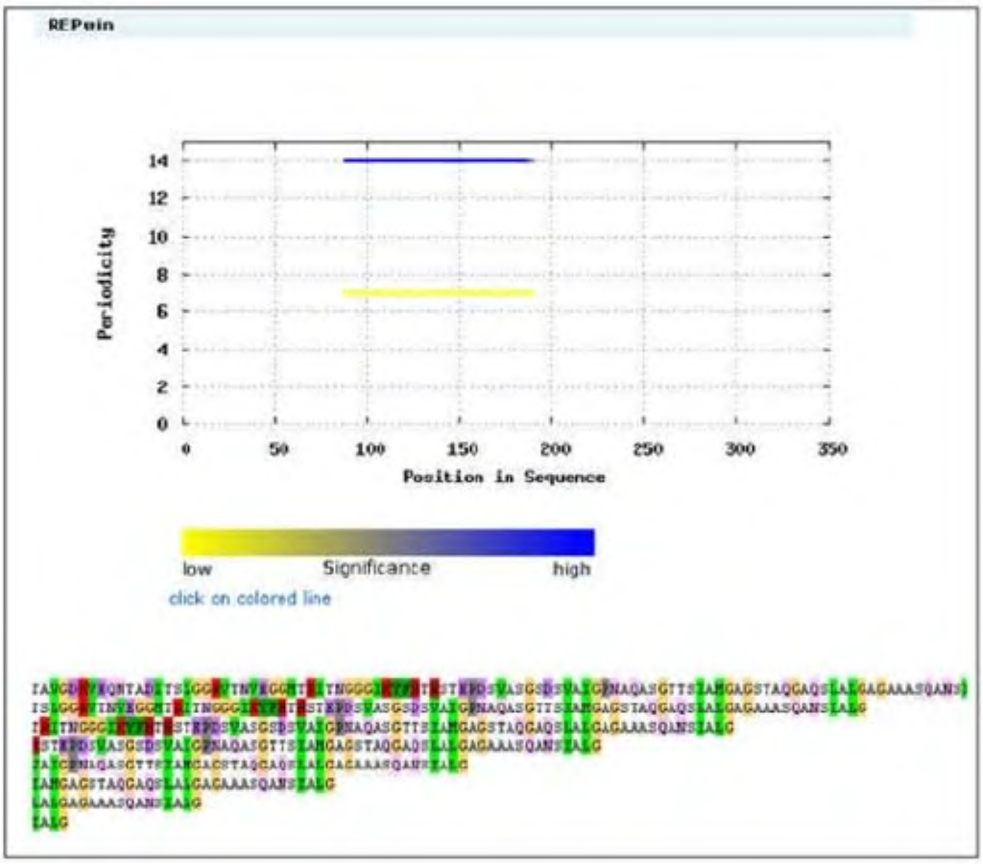

Figure 4. Panel depicting the results of REPPER for part of the hypothetical protein from *Y. pestis*.

identical repeats containing 64 and 248 amino acid residues respectively. The same sequence was submitted to the existing algorithms. The servers RADAR, TRUST and Internal repeat finder did not accept the input sequence due to the limitations in the number of amino acid residues. Further, REPPER server did not produce any significant results. In addition, there was a problem in obtaining results from REP and REPRO due to internal server error. The algorithm REPuter was not used for this sequence, since it has been developed for nucleotide sequences.

Since the algorithms RADAR, TRUST and Internal repeat finder have limitations in the number of amino acid residues given in the input, the protein fragment from residue numbers 251 to 575 (325 amino acid residues) of the hypothetical protein YPN_1947 from *Y. pestis* Nepal516 (>gi|108812109|ref|YP_647876.1|) was given as input. TRUST yielded the first 159 residues as a repeat, occurring in succession and hence allowing for mismatches (Figure 3). REPwin in REPPER resulted in composition-based repeats (Figure 4). RADAR produced gapped repeats, also allowing a certain degree of mismatch in the repeats. It also generated sequences which are subsets of a larger repeat, hence resulting in redundancy (Figure 5). In contrast, IdentSeek, the computing engine implementing the algorithm FAIR gave 11 non-redundant identical sequence repeats (Figure 6). In contrast to FAIR, which takes one or more sequences simultaneously, the commercially available Sequence Analysis Software Package of the University of Wisconsin Genetics Computer Group[30] (UWGCG) 'repeat' function can be run for only one sequence at a time. Each time the function is invoked, the minimum window size, stringency and other relevant parameters must be set. Thus, utilization of this program requires some initial time for learning. In addition, the FASTA format files must be converted to GCG format using the function 'fromfasta'. Additional parsing of the output, which allows mismatches depending on the stringency criteria (Figure 7) is necessary to obtain the location of the repeats. In this particular case, it is clear that the default criteria are insufficient to find identical repeats as the protein fragment mentioned above yields 100 repeats of lengths starting from two residues, not all of which are identical. On the other hand, although BioSuite[31] has a repeat analysis function in the pattern identification and matching sub-menu, this function does not appear to work for a large number of proteins or for protein sequences containing a large number of amino

```
-------------------------------------------------------------------------------------------
No. of Repeats|Total Score|Length |Diagonal| BW-From| BW-To| Level
 6| 405.16| 47| 54| 35| 81| 1
-------------------------------------------------------------------------------------------
 3- 59 (67.47/27.52)        AG....SqAHGSQSL.ALGAGATASQAnsialgassvTTVGAESDYSAYGLTA...PQTSVGEVG
 60- 112 (71.40/29.53)      MG....T.AQGNRKITGVAAGSADYDV....vnvaqlTAVGDKVEQNTADITS...LGGRVTNVE
 113- 161 (64.12/25.79)     GGmtriT.NGGG..IKYFHTHSTEPD.............SVASGSDSVAIGPNAqasGTTSIA.MG
 162- 218 (68.96/28.28)     AG...sT.AQGAQSL.ALGAGAAASQAnsialgassvTTVGAESDYSAYGLTA...PQTSVGEVG
 219- 271 (72.98/30.34)     VG....T.AQGNRKITGVAAGSADYDA....vnvaqlTAVGDKVDQNTADITS...LDGRVTNVE
 272- 320 (60.23/23.80)     GEmasiT.NGGG..VKYFHTHSTESD.............SVASGSDSVAIGPNAqasGTASVA.SG
-------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------
No. of Repeats|Total Score|Length |Diagonal| BW-From| BW-To| Level
 2| 39.07| 10| 157| 25| 34| 2
-------------------------------------------------------------------------------------------
 25- 34 (19.54/11.20)       NSIALGASSV
 184- 193 (19.54/11.20)     NSIALGASSV
-------------------------------------------------------------------------------------------
```

Figure 5. Results provided by the server, RADAR for part of the hypothetical protein from *Y. pestis*.

```
Total number of sequences uploaded = 1
Number of sequences having repeats = 1
**************************************************************************
>seq1
LGAGSQAHGSQSLALGAGATASQANSIALGASSVTTVGAESDYSAYGLTAPQTSVGEVGMGTAQGNRKI
TGVAAGSADYDVVNVAQLTAVGDKVEQNTADITSLGGRVTNVEGGMTRITNGGGIKYFHTHSTEPDSVA
SGSDSVAIGPNAQASGTTSIAMGAGSTAQGAQSLALGAGAAASQANSIALGASSVTTVGAESDYSAYGL
TAPQTSVGEVGVGTAQGNRKITGVAAGSADYDAVNVAQLTAVGDKVDQNTADITSLDGRVTNVEGEMAS
ITNGGGVKYFHTHSTESDSVASGSDSVAIGPNAQASGTASVASGKGTLA
Number of residues in the repeat = 39
ASQANSIALGASSVTTVGAESDYSAYGLTAPQTSVGEVG
[ 21 to 59 ]
[ 180 to 218 ]
--------------------------------------------------------------------------
Number of residues in the repeat = 21
DSVASGSDSVAIGPNAQASGT
[ 135 to 155 ]
[ 294 to 314 ]
--------------------------------------------------------------------------
Number of residues in the repeat = 8
GRVTNVEG
[ 106 to 113 ]
[ 265 to 272 ]
--------------------------------------------------------------------------
Number of residues in the repeat = 20
GTAQGNRKITGVAAGSADYD
[ 61 to 80 ]
[ 220 to 239 ]
--------------------------------------------------------------------------
Number of residues in the repeat = 6
ITNGGG
[ 118 to 123 ]
[ 277 to 282 ]
--------------------------------------------------------------------------
Number of residues in the repeat = 9
KYFHTHSTE
[ 125 to 133 ]
[ 284 to 292 ]
--------------------------------------------------------------------------
Number of residues in the repeat = 9
QNTADITSL
[ 96 to 104 ]
[ 255 to 263 ]
--------------------------------------------------------------------------
Number of residues in the repeat = 9
QSLALGAGA
[ 11 to 19 ]
[ 170 to 178 ]
--------------------------------------------------------------------------
Number of residues in the repeat = 5
SDSVA
[ 141 to 145 ]
[ 293 to 297 ]
[ 300 to 304 ]
--------------------------------------------------------------------------
Number of residues in the repeat = 5
SVASG
[ 136 to 140 ]
[ 295 to 299 ]
[ 316 to 320 ]
--------------------------------------------------------------------------
Number of residues in the repeat = 13
VNVAQLTAVGDKV
[ 82 to 94 ]
[ 241 to 253 ]
--------------------------------------------------------------------------
**************************************************************************
```

Figure 6. Output display of the proposed algorithm, FAIR for the same 325 residue protein fragment from *Y. pestis*.

```
Window: 3  Stringency: 10  Range: 50
Repeats: 100  April 23, 2008 12:39  .

       1 LGAG 4
         ||||       4 20
      15 LGAG 18

       1 LGA 3
         |||       3 14
      29 LGA 31

       1 LGA 3
          ||       3 11
      37 VGA 39

       4 GSQ 6
         |||       3 15
       9 GSQ 11

       5 SQA 7
         |||       3 13
      22 SQA 24

       7 AHG 9
         |||       3 12
      45 AYG 47

      10 SQ 11
         ||       2 9
      22 SQ 23

      12 SLALGA 17
         ||||||       6 24
      26 SIALGA 31

      16 GA 17
         ||       2 10
      38 GA 39
```

Figure 7. Output of UWGCG for the 325 residue protein fragment from *Y. pestis*. Only part of the output is shown for clarity.

acid residues (e.g. Titin). Furthermore, although it gives the location of the repeat like FAIR, with the default parameters, it found only three repeats in the 325 residue fragment (Figure 8).

Furthermore, FAIR (and thus, IdentSeek) can be used to perform analysis on both protein and nucleotide sequences, whereas the other tools can only work for either

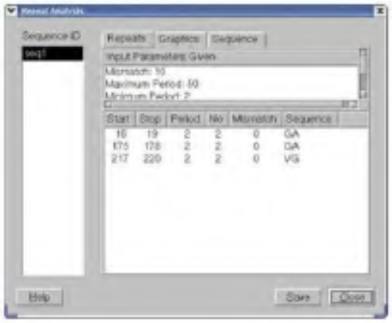

Figure 8. Output of BioSuite for the 325 residue protein fragment from *Y. pestis*.

```
>gi|51593846|ref|NC_006153.1| Yersinia pseudotuberculosis IP 32953 plasmid
pYV, complete sequence
GGGGGCACTTGTCACATCCATTCCCGCTCCAACCGGTTCAGTCGCTCGGCGATGGTCGTCAGCTCTCGCA
ACTTTTCTCTCTGTTCAACCAAATGCTCATCCATTAAACTGCTTCGTCCAAGTTGAATCAGTAGTGGAAT
AGAGTCCACTGTAAAGTTTCGCATATCATCAGTAAACTCCGATACCTCCTGTAGCGTAATAGCTGCATGG
GCCGCCATCATCTGACGGTGAATTGCGGTATACTGCTCAGTGCCGAATTTCACTACAGGCCGGGTACTGT
.....
.....

Number of residues in the repeat = 219
CCATTCACCGATATCTCGTACCGCATGATGGACATCCATATAGCCCC
CTACAGGCAGCCATTCACTTTTCAGGCTTCGGAACACTCTTTCCATT
GGTGAGTTATCCAGGCAGTTACCCCTGCGGCTCATACTCTGCATCAC
TCCGGTCCTCCACAGTAACTGCCTGGATTTCTTACTCCTGTACTGCC
CTCCCTGATCCGAATGAAACAGCAGCCTCTT
[ 3018 to 3236 ]
[ 26108 to 26326 ]
--------------------------------------------------------------------------

Number of residues in the repeat = 126
CCCCATCAAAAAATTGGTAAACGCGGCCGGGGATTTCAAACGGCCAA
ACCGTGCCTGGTCAACGCTTCAGGGATTTGAAACATTGCGACGACTG
AATAAAGGTCAATTTGATATTTGGTTACGCCG
[ 31355 to 31480 ]
[ 35938 to 36063 ]
--------------------------------------------------------------------------

Number of residues in the repeat = 143
CTCTGCGGTAATTCTGGCAGATCAGTTAAATCATTTTCTCTGACATG
AAGTTTTTTCAGGGAAGGGGGTAAATCGGGTAATGTTTCCAGTAAAT
TGTTATCAGCATAAATCGCAGCCAAGAAGGGCAAGTTTTGCAACTCT
GG
[ 33629 to 33771 ]
[ 34121 to 34263 ]
--------------------------------------------------------------------------

Number of residues in the repeat = 434
GAATGTCTGGCGCTCGAGGTCGGTCAGGGGTTACGTGGAGATGATGT
TGTGGCTGTCATGGACAGATTAAAACATTCGCTGGGGCGTATTCCAC
AAAGGCTGCAGACAGATAACGGCAGCGAATTCATCTCGAAGTCGATG
GACCGATGGGCGTATGAAAACAGGGTCACGATGGACTTTTCACGCCC
CGGAAAGCCTACAGATAATGCCTTTATCGAGTCATTTAATGGCAGTC
TGAGGGATGAATGTCTGAACGTGCACGGGTTCCTTTCTCTGGAAGAT
GCTCAGGAGAAAATTGAACAATGGCGGCAAGAATATAATCATTTTCG
TCCGCATTCTTCGTTAAATAACCTGACTCCGGCAGAATTTGCCCGAA
GTCATCAAAAAGGTCCGGATCTCTGATTTAGCCTGGTACGGATATTG
GGGAGGGATCA
[ 16165 to 16598 ]
[ 66289 to 66722 ]
--------------------------------------------------------------------------

Number of residues in the repeat = 542
GAGTAAAGGTCAATCACCAGTGCCAGGTAGCACCAGCCACCATTGAC
TTTGATATAGCTGATATATCCACTCCACACACGATTGGGCTCCGACG
GCTTAAACTGCCTTTTCAGTAAGTCTGGCGATGCCAGTGACACCTCC
CGTTCGCCACGGTAACGAGGTTTTCCGGGTTGGCGACTCGCCAGACC
ACATTCCCGCATCAGCCTGCCAGCCAGCCAGCGACCCACGTTATAAC
CCGACTGACGCATCAGCAGACTCAGTGTTCTGTTGCCGGCTGCTCCC
CGGCTCTGTTGATGCAATTCACGCAGTTGCAGACGTTTTACATCAAC
CCGCAAATTCAGCGAAGCAGAGTAAACACTGCGCGTTATTTTGAGCA
GGCGGCACAATTCGACCACTGGCCATTTTGTTTTCAGCCGTGTGATT
AACGCAAAGATTTGATGGGGAACTCGCTCATCAACACGGCTGCCTGC
TTTAGTATTTCTTTTTCCATTTCCAGCCGTTTAATCTGCGCCCTGAG
CGACTGGATTTCACGTTGCTCTGGT
[ 7068 to 7609 ]
[ 26421 to 26962 ]
--------------------------------------------------------------------------

Number of residues in the repeat = 111
GCTCAATAGCTTCCAGTTTGAATTCAGGAGAAAAACGGCGTTTGGTC
TGTTTAGGTTGAGTCATTATCAATTACCTCAATTGGCTGTCATTAAA
TTAACAGGCTATTGAGG
[ 7743 to 7853 ]
[ 27097 to 27207 ]
--------------------------------------------------------------------------

Number of residues in the repeat = 265
GGGGGTAAATCGGGTAATGTTTCCAGTAAATTGTTATCAGCATAAAT
CGCAGCCAAGAAGGGCAAGTTTTGCAACTCTGGCAATTCTTCCAGCT
TATTACTACTAGCAGTAAGATATTCCAGTGAAGGAGGTAAATCGGAT
AATGCCTTCAGATTGTTATTATCAACTCGAAGTGATTCCAGGCTCTG
CGGTAATTCTGGCAGATCAGTTAAATCATTTTCTCTGACATGAAGTT
TTTTCAGGGAAGGGGGTAAATCGGGTAATG
[ 33938 to 34202 ]
[ 34184 to 34448 ]
--------------------------------------------------------------------------

Number of residues in the repeat = 116
TGTAGTGGATCACAAAATCAGGACACCTTCGTAGCCTTTTTCCCCTG
CTCTTCGTATTCACAGGGCGGTAATCCACCATTGTGCCGATGGGGGT
GTGTTGTAATAACTTTGTATCC
[ 2902 to 3017 ]
[ 25994 to 26109 ]
```

Figure 9. Part of the output of FAIR for the entire plasmid sequence from *Yersinia pseudotuberculosis*, showing several large repeats.

```
--------------------------------------------------------------------------
# 68526 -3 100 reputer_bibiserv_1168227935_1555652638.xmlrpc
--------------------------------------------------------------------------
676 7067        F       676 26420 -3    0.00e+00        **
558 16040       F       558 66164 -3    0.00e+00
522 16077       F       522 66201 -3    0.00e+00
474 16124       F       474 66248 -1    7.89e-274
434 16164       F       434 66288 0     6.71e-253       *
265 33937       F       265 34183 0     3.76e-151       *
260 33628       F       260 34120 -3    3.01e-140       **
219 3017        F       219 26107 0     1.86e-123       *
197 33691       F       197 33937 -3    1.11e-102
193 6640        F       193 25993 -3    2.67e-100
172 22716       F       172 22806 -3    8.29e-88
147 6686        F       147 26039 -1    1.83e-77
131 7610        F       131 26963 -1    7.00e-68
126 31354       F       126 35937 0     1.82e-67        *
137 15986       F       137 66110 -3    4.92e-67
137 22761       F       137 22851 -3    4.92e-67
116 2901        F       116 25993 0     1.91e-61        *
127 22806       F       127 22851 -3    4.11e-61
111 7742        F       111 27096 0     1.96e-58        *
117 2900        F       117 6639 -2     2.92e-57
116 33772       F       116 34264 -2    1.15e-56
116 33772       F       116 34018 -2    1.15e-56
116 22712       F       116 22847 -3    1.31e-54
107 22798       F       107 22888 -3    2.69e-49
104 22739       F       104 22874 -3    1.58e-47
--------------------------------------------------------------------------
```

Figure 10. Output of REPuter for the nucleotide sequence, a plasmid from *Y. pseudotuberculosis*.

protein or nucleotide sequences. The proposed algorithm was also tested for nucleotide sequences using the plasmid pYV from *Yersinia pseudotuberculosis* IP32953 (gi|51593846 |ref| NC_006153.1) in FASTA format. As the input sequence contains 68,532 nucleotides, only a part of the input sequence is shown in Figure 9 and the minimum number of nucleotides per repeat was set as greater than or equal to 100. The program produced eight independent identical repeats containing between 116 and 542 nucleotides. REPuter, the only freely available program for finding repeats in nucleotide sequences (other than FAIR), displayed six unique repeats (marked as * in Figure 10), similar to that of the repeats produced by FAIR. However, the REPuter server did not show the remaining two repeats (marked as ** in Figure 10) containing 542 and 142 nucleotides as produced by FAIR, as it displayed them (marked as ** in Figure 10) with mismatches, which are denoted by negative scores, whereas the exact repeats are denoted by a score 0. Since the proposed algorithm is designed to show only identical repeats, the program FAIR produces eight unique identical repeats.

Conclusion

The proposed algorithm identifies all the identical sequence repeats in both protein and nucleotide sequences and produces a comprehensive output by displaying the number of repeats with their start and end positions. Due to dynamic allocation of memory, the memory required for the program is exactly equal to the memory required to store the repeats and thus, the algorithm is more effi-

cient in dealing with sequences having a large number of amino acid residues. This algorithm is unique in its utilization of dynamic programing rather than local alignment and in its speed compared to other algorithms. Furthermore, the algorithm has been implemented as a web-based computing engine, IdentSeek, designed for giving the user a precise, clear and usable output. This would help in the analysis of protein sequences, especially to facilitate the study of evolutionary history of proteins and diseases related to genes and proteins. The computing engine is available on the world wide web at http://bioserver1.iisc.ernet.in/ident/.

1. Marcotte, E. M., Pellegrini, M., Yeates, T. O. and Eisenberg, D., A census of protein repeats. *J. Mol. Biol.*, 1999, **293**, 151–160.
2. Mojica, F. J. M., Diez-Villasenor, C., Soria, E. and Juez, G., Biological significance of a family of regularly spaced repeats in the genomes of archaea, bacteria and mitochondria. *Mol. Microbiol.*, 2000, **36**, 244.
3. Fernando, S. A., Selvarani, P., Das, S., Kiran Kumar, Ch., Mondal, S., Ramakumar, S. and Sekar, K., THGS: A web-based database of transmembrane helices in genome sequences. *Nucleic Acids Res.*, 2004, **32**, D125–D128.
4. Arai, M., Ikeda, M. and Shimizu, T., Comprehensive analysis of transmembrane topologies in prokaryotic genomes. *Gene*, 2003, **304**, 77–86.
5. Saier Jr, M. H., Tracing pathways of transport protein evolution. *Mol. Microbiol.*, 2003, **48**, 1145–1156.
6. Saaf, A., Baars, L. and Von Heijne, G., The internal repeats in the Na^+/Ca^{2+} exchanger-related *Escherichia coli* protein YrbG have opposite membrane topologies. *J. Biol. Chem.*, 2001, **276**, 18905–18907.
7. Taylor, E. W. and Agarwal, A., Sequence homology between bacteriorhodopsin and G-protein coupled receptors: Exon shuffling or evolution by duplication? *FEBS Lett.*, 1993, **325**, 161–166.
8. Andrade, M. A., Perez-Iratxeta, C. and Ponting, C., Protein repeats: Structures, functions, and evolution. *J. Struct. Biol.*, 2001, **134**, 117–131.
9. Kachroo, P., Ahuja, M., Leong, S. A. and Chattoo, B. B., Organization and molecular analysis of repeated DNA sequences in the rice blast fungus *Magnaporthe grisea*. *Curr. Genet.*, 1997, **31**, 361–369.
10. Heringa, J., Detection of internal repeats: How common are they? *Curr. Opin. Struct. Biol.*, 1998, **8**, 338–345.
11. Koonin, E. V., Mushegian, A. R., Galperin, M. Y. and Walker, D. R., Comparison of archeal and bacterial genomes: Computer analysis of protein sequence predicts novel function and suggests chimeric origins for the archaea. *Mol. Microbiol.*, 1997, **25**, 619–637.
12. Djian, P., Evolution of simple repeats in DNA and their relation to human diseases. *Cell*, 1998, **94**, 155–160.
13. Benvenga, S., Campenni, A. and Facchiano, A., Internal repeats of prion protein and A beta PP, and reciprocal similarity with the amyloid-related proteins. *Amyloid*, 1999, **6**, 250–255.
14. Perutz, M. F., Glutamine repeats and neurodegenerative diseases: Molecular aspects. *Trends Biochem. Sci.*, 1999, **24**, 58–63.
15. Van de Lagemaat, L. N., Gagnier, L., Medstrand, P. and Mager, D. L., Genomic deletions and precise removal of transposable elements mediated by short identical DNA segments in primates. *Genome Res.*, 2005, **15**, 1243–1249.
16. Wu, T. T., Miller, M. R., Perry, H. M. and Kabat, E. A., Long identical repeats in the mouse gamma 2b switch region and their implications for the mechanism of class switching. *EMBO J.*, 1984, **3**, 2033–2040.
17. Bertin, J. *et al.*, Human CARD4 protein is a novel CED-4/Apaf-1 cell death family member that activates NF-B. *J. Biol. Chem.*, 1999, **274**, 12955–12958.
18. Krych-Goldberg, M., Hauhart, R. E., Subramanian, V. B., Yurcisin, II B. M., Crimmins, D. L., Hourcade, D. E. and Atkinson, J. P., Decay accelerating activity of complement receptor type 1 (CD35): Two active sites are required for dissociating C5 convertases. *J. Biol. Chem.*, 1999, **274**, 31160–31168.
19. McAllister-Luca, L. M. *et al.*, An essential aspartic acid at each of the two allosteric cGMP binding sites of a cGMP specific phosphodiesterase. *J. Biol. Chem.*, 1995, **270**, 30671–30679.
20. Blackburn, E. H., Switching and signaling at the telomere. *Cell*, 2001, **106**, 661–673.
21. Söding, J., Remmert, M. and Biegert, A., HHrep: *de novo* repeat detection and the origin of TIM barrels. *Nucleic Acids Res.*, 2006, **34**, W137–W142.
22. Schultz, J., Milpetz, F., Bork, P. and Ponting, C. P., SMART, a simple modular architecture research tool: Identification of signalling domains. *Proc. Natl. Acad. Sci. USA*, 1998, **95**, 5857–5864.
23. Andrade, M. A., Ponting, C. P., Gibson, T. J. and Bork, P., Homology-based method for identification of protein repeats using statistical significance estimates. *J. Mol. Biol.*, 2000, **298**, 521–537.
24. Gruber, M., Soding, J. and Lupas, A. N., REPPER – Repeats and their periodicities in fibrous proteins. *Nucleic Acids Res.*, 2005, **33**, W239–W243.
25. Heringa, J. and Argos, P., A method to recognize distant repeats in protein sequences. *Proteins*, 1993, **17**, 391–411.
26. Heger, A. and Holm, L., Rapid automatic detection and alignment of repeats in protein sequences. *Proteins*, 2000, **41**, 224–237.
27. Szklarczyk, R. and Heringa, J., Tracking repeats using significance and transitivity. *Bioinformatics*, 2004, **20**, i311–i317.
28. Kurtz, S., Choudhuri, J. V., Ohlebusch, E., Schleiermacher, C., Stoye, J. and Giegerich, R., REPuter: The manifold applications of repeat analysis on a genomic scale. *Nucleic Acids. Res.*, 2001, **29**, 4633–4642.
29. Pellegrini, M., Marcotte, E. M. and Yeates, T. O., A fast algorithm for genome-wide analysis of proteins with repeated sequences. *Proteins*, 1999, **35**, 440–446.
30. Devereux, J., Haeberli, P. and Smithies, O., A comprehensive set of sequence analysis programs for the VAX. *Nucleic Acids Res.*, 1984, **12**, 387–395.
31. NMITLI-BioSuite team, BioSuite: A comprehensive bioinformatics software package (A unique industry–academia collaboration). *Curr. Sci.*, 2007, **92**, 29–38.

ACKNOWLEDGEMENTS. We acknowledge the use of the Bioinformatics Centre, the Interactive Graphics Based Molecular Modeling facility and the Supercomputer Education and Research Centre at IISc. We thank the Department of Information Technology for funding this project. Part of this work is supported by the Department of Biotechnology sponsored Institute wide computational biology programme.

Received 16 November 2007; revised accepted 11 June 2008