Filtering Redundancies For Sequence Similarity Search Programs

http://www.jbsdonline.com

Abstract

Database scanning programs such as BLAST and FASTA are used nowadays by most biologists for the post-genomic processing of DNA or protein sequence information (in particular to retrieve the structure/function of uncharacterized proteins). Unfortunately, their results can be polluted by identical alignments (called redundancies) coming from the same protein or DNA sequences present in different entries of the database. This makes the efficient use of the listed alignments difficult. Pretreatment of databases has been proposed to suppress strictly identical entries. However, there still remain many identical alignments since redundancies may occur locally for entries corresponding to various fragments of the same sequence or for entries corresponding to very homologous sequences but differing at the level of a few residues such as ortholog proteins. In the present work, we show that redundant alignments can be indeed numerous even when working with a pretreated non-redundant data bank, going as high as 60% of the output results according to the query and the bank. Therefore the accuracy and the efficiency of the post-genomic work will be greatly increased if these redundancies are removed. To solve this up to now unaddressed problem, we have developed an algorithm that allows for the efficient and safe suppression of all the redundancies with no loss of information. This algorithm is based on various filtering steps that we describe here in the context of the Automat similarity search program, and such an algorithm should also be added to the other similarity search programs (BLAST, FASTA, etc…).

Introduction

Several programs have been developed to rapidly scan a database in order to find the alignments matching a query sequence, and thus try and derive its biological function. Among them, FASTA (1), BLAST (2, 3) and AUTOMAT (4, 5), are quite powerful. Often, in the resulting output listing, one may find several times an alignment involving the same sequence, but originating from different banks. One way to avoid this kind of redundancy is the use of non-redundant databases, constructed by gathering the various banks in order to have the widest available information, and by suppressing the exactly identical entries to avoid double hits. Among such banks, one can for instance cite OWL (6), KIND (7), or the NCBI non redundant database, NRDB (at URL http://www.ncbi.nlm.nih.gov). Nevertheless, even with such non-redundant data banks, redundancies can still occur involving homologous sequences that are slightly divergent (such as ortholog proteins or viral proteins from various strains which slightly diverge from each other). Since the target sequences that interest the biologist are usually in the range of 20-30% identity with the query sequence (otherwise it would be trivial), these alignments correspond to fragments and not complete sequences. They might involve closely linked target proteins that often produce identical fragments, thus executing a local redundancy. As the number of sorted sequences in the output listing is limited for ease of understanding, these redundancies happen at the expense of the availability of more information for the biologist.

Hubert Cantalloube1,2
Jacques Chomilier3,*
Sylvain Chiusa1,3
Mathieu Lonquety3
Jean-Louis Spadoni4
Jean-François Zagury1,4,**

1Groupe Bioinformatique
Génomique et Traitement des Pathologies
du Système Immunitaire
INSERM EMI0355
15 rue de l’Ecole de Médecine
75006 Paris, France
2Département d’Électro-Magnétisme et
Radar
ONERA
chemin de la Hunière
91761 Palaiseau, France
3Laboratoire Minéralogie-
Cristallographie, CNRS
Universités Paris 6 and 7
Case 115
75252 Paris cedex 05, France
4Chaire de bioinformatique
CNAM, 292 rue Saint Martin
75003 Paris, France

*Email: jacques.chomilier@lmcp.jussieu.fr
**Email: jfz@ccr.jussieu.fr
Another approach has been to remove all highly identical sequences from the database by a clustering approach (8) that gives rise to an irreversible loss of information in the database. If this approach might be useful when looking for protein domains or folds, it is however not applicable to protein fragments similarities of smaller size such as epitopes, nor to DNA sequences.

In the present work, we show that even a non-redundant database such as the NCBI nrdb can still yield many redundant alignments, and we present an algorithm that is able to suppress local redundancies at the last stage of recovery, whatever database is being used. This algorithm was mentioned in our previous work (5), however such an algorithm has still not been implemented in the other similarity search software available for biologists. It is based on various filtering steps and we will describe it in the context of Automat, a sequence similarity search program previously shown to be as powerful and complementary to BLAST for searches on protein sequences (4, 5).

**Algorithm**

In the phase of retrieval of the similarities between the query and target sequences from the database, all similarity search programs quantify the alignments found in order to sort them. This is generally done through a homology matrix (9) that gives, for any pair of nucleic acids or amino acids, a cost for the substitution of one element of the pair by the other.

The method for suppressing the redundant alignments involving related target sequences from the database is based on a series of filtering steps that discriminates truly identical alignments. These filtering steps have been chosen in order of decreasing simplicity to optimize the speed of the sorting process:

I. The alignments generated by the program are rescored using a homology matrix that is slightly modified from the usual one so that no simple rational relationship exists between the differing values in the matrix. The initial homology matrix is slightly modified by adding to each entry a randomly chosen small number, typically of the $10^{-6}$ level. Such small cost variations will only possibly swap the order of consecutive alignments in a non-significant way compared to the unperturbed matrix sorting. With this perturbed matrix, if two alignments yield an identical score, it means there is a very high probability that the residues and substitutions involved in these two alignments are identical. Nevertheless they could be placed at different positions along the sequences.

II. The second criterium to discriminate alignments of identical scores after the previous step is to identify the length of the aligned fragments. If both length and score are equal, then they are differentiated in the following step.

III. The third criterium looks for the first position in the query of the alignment that is remaining from the second step. In case of two equal first positions, one proceeds to the next step.

IV. A hash-value is then applied to the alignment, to take into consideration the order of the residues. The hash value $y$ is computed recursively position by position as:

$$y_n = a_n + Ky_{n-1}$$

where $a_n$ is the symbol code for the $n^{th}$ position of the target sequence in the alignment, and $K$ is the number of symbol codes (namely 20 for
amino acids and 4 for nucleic acids). \( Y \) is computed using an integer variable, silently overflowing in case of a long alignment.

V. If two alignments still have identical hash-values, they will most often (if not always) be identical, i.e., correspond to a redundancy. However, in that case, and for more reliability, the program explicitly compares the two sequences and in cases of differing residues, sorts them by alphabetical order. The comparison is performed in the fragments of sequences involved in the alignments with the query. This step, which is fairly computer time consuming, is performed only on a very limited number of sequences.

VI. Sequences remaining in this last step are definitely considered as redundant, and one still has to decide which one to leave in the output listing and, consequently, which one to mark as a redundancy. For this, the algorithm deals with the full sequences and sorts first the longest sequence. In cases of equal lengths, the longest header description is ranked first.

Non-redundant alignments will be discriminated by this algorithm at any of the five first steps.

Implementation

This algorithm has been implemented in the Automat database-scanning program. Automat has already been described (4, 5) and briefly it functions as follows. The query sequence is split into words of minimal length \( w \) that is selected by the user (typically \( w=3 \) for proteins), and an alphabet of classes of equivalent amino acids can also be defined. Hits of these words (called triggers for Automat) with identical words in the target sequence are collected, and then the program gathers the neighboring triggers which have the same offset between the query and the target and which are at a distance of less than \( k \) – mismatching – residues (by default \( k=20 \)) in order to generate alignments. Alignments adding up to \( n \) or more matching residues (typically \( n=8 \) for proteins) are qualified for a scoring with the homology matrix. In Automat, the score corresponds to the maximum cumulated cost of aligned symbols found within the generated alignment.

For trigger detection, Automat relies on a finite state automation, and triggers are stored in an offset-hashed chained-list structure. Once the target sequence is scanned, triggers are clustered and scored into a heap structure (semi-sorted binary tree), the dimension of which is the maximum number of reported alignments per sequence. Matching sequences are themselves stored in a larger heap structure according to their maximum alignment score, with the dimension of that heap being the maximum number of reported sequences.

Once the full database is scanned and the number of alignments with the best scores has been selected, the methodology for removing local redundancies can be applied during the heap readout procedure (the hash value is systematically computed, but the target sequence is retrieved, and the alphabetic ordering comparison is made only if two candidates for promotion to the top of the heap share the same values for both scores (these are most often redundant sequences).

Additionally, we have set a function to keep, in memory, the removed alignments and to let them appear on the screen (but dimmed in intensity) by a simple mouse click. This function allowed us to confirm that all the removed alignments were effectively redundancies. We also evaluated the time consumed for the removal of the redundant alignments, and it was negligible, typically less than the accuracy of the system-process elapsed-time measurement. This is understandable since
the redundancy suppression algorithm applies to the few hundreds or thousands of sequences with the best scores that are selected by the programs at the request of the user, and the time-consuming 2 by 2-alphabetic comparisons between target sequences have now been limited.

Programs such as BLAST or FASTA also generate a list of alignments sorted by scores and it is possible to add the redundancy suppression algorithm as an internal module if one has access to the source code, or as an external module if one wants to treat externally the output files generated by these programs. For Automat, we in fact sidestep the rescoring of step 1 by using directly the pertubated matrix for generating the output alignments. This could also be done for the other programs if the algorithm is added as an internal module: if these programs require integer matrices, one can simply multiply the pertubated matrix by $10^6$ to transform it into integers. In cases when the algorithm is added as an external program module, it could be advantageous to put the rescoring of step 1 after step 3 since it is slightly more costly in time. However due to the generally limited number of alignments to sort (less than a few thousands) and the speed of the algorithm, it would not change much practically.

**Results**

In order to evaluate the level of local redundancies and the efficiency of their removal, we will present some examples with proteins in Table I. We used the standard parameters for Automat with an alphabet composed of seven classes (VILM, FYW, RQE, HSD, ACT, GN and P), the three parameters w/n/k equal to 3/8/20 (default values) and the Blosum62 matrix (10).

The first example can be considered as having a low level of redundancy. It is taken from the work of Berezovsky and Trifonov (11-13) where they extracted from 23 complete bacterial genomes prototype sequences of around 30 amino acids long, with a maximum divergence of 10 positions over the fragment, *i.e.*, around 30%. The aleph prototype, given in Table I, has been searched for in the complete eucaryote genome of *A. Thaliana* (14), to check if one such prototype could also be found in a vegetal genome. The answer was yes, and the level of redundancy was very low, around 11.4% with 500 output sequences having been sorted. The second example dealt with fragments of the gp120 envelope protein from Human Immunodeficiency Virus type 1 (HIV-1), known to present a high degree of vari-

### Table I

<table>
<thead>
<tr>
<th>Name</th>
<th>Sequence</th>
<th>Size</th>
<th>Output</th>
<th>Nb RED</th>
<th>% RED</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aleph</td>
<td>GEIVALVGFSGGKGSTLLRALAGGLKPDGG</td>
<td>30</td>
<td>500</td>
<td>229</td>
<td>4.6</td>
</tr>
<tr>
<td>gp120</td>
<td>LKCTDLKNDTNTNNSSSGRMIMEKGEIKNCSFNISTS</td>
<td>100</td>
<td>500</td>
<td>1563</td>
<td>31.3</td>
</tr>
<tr>
<td>AA101-200</td>
<td>IGGVKVKEAFFYLIDIPINDTTSYKLYSTCNSTSV</td>
<td>100</td>
<td>500</td>
<td>172</td>
<td>34.4</td>
</tr>
<tr>
<td>gp120</td>
<td>LDIDIPINDTTSYKLYSTCNSTSVTQACPVSFEP1PHYCAPAGFAILKC</td>
<td>50</td>
<td>500</td>
<td>2108</td>
<td>42.2</td>
</tr>
<tr>
<td>AA151-200</td>
<td>ITIQACPVSFEP1PHYCAPAGFAILKC</td>
<td>50</td>
<td>500</td>
<td>307</td>
<td>61.4</td>
</tr>
<tr>
<td>Hydrolase</td>
<td>KPANVPDKFYDAAFSDSWNALQVPSNWQCHGDFRPIYTTNVPFPPNDFPYYPEDNPTGCYRTFQIPEKWKDRRILLHFEAVDSAFWNINGNFVGYSQ</td>
<td>100</td>
<td>500</td>
<td>1561</td>
<td>31.2</td>
</tr>
<tr>
<td>AA101-200</td>
<td>KPYTPFPDNPTGCYRTFQIPEKWKDRRILLHFEAVDSAFWNINGNFVGYSQ</td>
<td>50</td>
<td>500</td>
<td>1776</td>
<td>35.5</td>
</tr>
<tr>
<td>Hydrolase</td>
<td>FFASWNINGNFVGYSQ</td>
<td>50</td>
<td>500</td>
<td>167</td>
<td>33.4</td>
</tr>
</tbody>
</table>
ability among closely related strains. Although we used the non-redundant NCBI nrdb database, the level of local redundancy detected ranged from 31.3% to 61.4% according to the query and the number of alignments sorted. The last example dealt with a bacterial hydrolase from *A. Thaliana* and with the NCI nrdb database, with the level of local redundancy being between 29.0% and 35.5% according to the query and the number of alignments sorted.

Table II presents the sorting efficacy of each step of the algorithm. It has been evaluated for the five sequences of Table I in the case of 500 alignments generated in the output listing. The numbers correspond to the percentage of sequences (relative to the number of sequences requested in the output listing) still having an identical score after the first step, or a same fragment length after the second step, *et cetera*. In other words, the difference of percentage with the preceding step represents the fraction of sequences that have been treated and accepted as non-redundant. The last line of Table II corresponds to the percentage of effective redundancies which are given in Table I.

<table>
<thead>
<tr>
<th>Step of the algorithm:</th>
<th>Aleph</th>
<th>Gp120-100</th>
<th>Gp120-50</th>
<th>Hydrolase-100</th>
<th>Hydrolase-50</th>
</tr>
</thead>
<tbody>
<tr>
<td>0. Sequences initially selected by the program (unperturbed matrix)</td>
<td>448 (89.6)</td>
<td>425 (85.0)</td>
<td>475 (95.0)</td>
<td>391 (78.2)</td>
<td>420 (84.0)</td>
</tr>
<tr>
<td>1. Sorting by identical score (perturbed matrix)</td>
<td>191 (38.2)</td>
<td>334 (66.8)</td>
<td>458 (91.6)</td>
<td>203 (40.6)</td>
<td>229 (45.8)</td>
</tr>
<tr>
<td>2. Sorting by fragment length</td>
<td>107 (21.4)</td>
<td>260 (52.0)</td>
<td>447 (89.4)</td>
<td>167 (33.4)</td>
<td>187 (37.4)</td>
</tr>
<tr>
<td>3. Sorting by first position of the query</td>
<td>94 (18.8)</td>
<td>259 (51.8)</td>
<td>446 (89.2)</td>
<td>158 (31.6)</td>
<td>180 (36.0)</td>
</tr>
<tr>
<td>4. Sorting by hash score</td>
<td>80 (16.0)</td>
<td>259 (51.8)</td>
<td>446 (89.2)</td>
<td>158 (31.6)</td>
<td>180 (36.0)</td>
</tr>
<tr>
<td>5. Sorting by alphabetical identity</td>
<td>57 (11.4)</td>
<td>172 (34.4)</td>
<td>307 (61.4)</td>
<td>145 (29.0)</td>
<td>167 (33.4)</td>
</tr>
</tbody>
</table>

As shown in Table II, all steps can have a significant impact on discriminating non-redundant alignments.

**Discussion**

We have presented a simple and efficient algorithm that filters local redundancies into the list of alignments retrieved from database scanning. Our methodology can be added simply to any database scanning program as shown here for Automat. Other programs have previously been developed to help filter alignments generated by BLAST, such as MSP Crunch (15). However, they do not address the problem of redundant alignments and cannot easily be integrated into the scanning program.

The first example of Table I is actually a consensus sequence of roughly 30 amino acids retrieved from 23 bacterial genomes, assumed to correspond to ancestor proteins. Therefore, as it is an “artificial” sequence, the level of redundancy was expected to be fairly small. In a full genome such as the one of Arabidopsis Thaliana (14), we found a local redundancy rate of about 11%. On sequences with a high level of local mutations, such as HIV sequences, the rate of redundancies on a non-redundant data bank such as the NCBI nrdb database could go as high as 61% depending on the size of the fragment studied. Our study underlines the considerable level of local redundancies even in a database supposed to be non-redundant.

Actually the residual level of redundancy is located in particular fragments of the sequences, and it appears difficult to further clean up a database, since one could...
lose valuable information. Removing local redundancies after a search scan is therefore much more suitable.

Cantalloube et al.

Studies based on widespread genome and gene/protein family analyses, which rely on the blind screening of databases, do not generally take into account the presence of redundancies even though we have just seen how important they can be. Removing alignments involving related target proteins should also be very useful for researchers interested in local similarities, for example immunologists looking for autoimmune epitopes (4). Whatever field is concerned, filtering redundancies will definitely render the output results of similarity search programs more accurate and convenient for all biologists.

The algorithm is implemented in the running version of the Automat server at the URL http://bioserv.rpbs.jussieu.fr/. The source code is available upon request to the corresponding authors.

References and Footnotes


Date Received: July 19, 2004

Communicated by the Editor Ramaswamy H Sarma