DegenRev: Degeneracy-Based Full Backtranslation Algorithm for Oligopeptide

Mohieddine Missaoui, David R.C. Hill, Pierre Peyret

DegenRev: Degeneracy-Based Full Backtranslation Algorithm for Oligopeptide

Mohieddine Missaoui[1, 2] **David R.C. Hill**[2]
Pierre Peyret[1]

Research Report LIMOS/RR-07-10

05 Juin 2007

[1] LBP – UMR CNRS 6023, Blaise Pascal University, Clermont-Ferrand, France
[2] LIMOS – UMR CNRS 6158, Blaise Pascal University, Clermont-Ferrand, France

[1] missaoui@isima.fr

Abstract

In order to design microarray oligonucleotides, in the context of new metabolic pathways discovery, it appears that a full backtranslation of oligopeptides is a promising approach. Protein to DNA reverse translation is a time-consuming task that can provide unreasonable quantities of data. This is why most current applications use genetic degenerated code or data mining-based techniques to select the best reverse translation of a short protein sequence called oligopeptide. When the purpose is only to design short oligos it is particularly interesting to have the complete sequences to solve the design problems of enzyme specific oligos for microarrays. In this paper, we revisit existing bioinformatics applications, which bring reverse translation solutions, and we present a new algorithm based on input oligopeptide degeneracy able to efficiently compute a full reverse translation. We propose an implementation with the C programming language and we show its performance statistics on simulated and real biological datasets.

Keywords: Degeneracy, Oligopeptide, Full backtranslation,

1 Introduction

Reverse translating a given peptide allows the determination of the nucleic sequences at its origin. This is a retro-engineering process, contrary to the natural translation of RNA (*Ribonucleic acid*) which gives a protein sequence that has specific activity in the studied organisms. Applications using reverse translation - known as *backtranslation* - introduce different solutions to get the more appropriate nucleic sequence among all possibilities. The resulting sequence is considered as the sequence which is at the origin of the studied protein. To perform this task one uses codon that associates a triplet of nucleic acids to each amino acid. Each amino acid can be represented by 1, 2, 3, 4 or 6 codons. Furthermore, the backtranslation problem appeared after the protein synthesis mechanism discovery [9] followed-up by the entire genetic code decoding in 1961. Nowdays, backtranslation is required in more and more applications using protein to DNA or RNA passing. For instance, Molecular Phylogeny is based on genetic code to perform a phylogenetic tree construction with sequence homology detection. Information held by proteins is useful when constructing multiple DNA alignments for phylogenetic analysis [15]. In this case, backtranslation is just a substitution of amino acid by the existing codon in the original nucleic sequence. Thus, backtranslate a protein to DNA can preserve this information. Nevertheless, genomic and transcriptomic databases are used as a source of sequences for protein alignments, so it is possible to find CDSs (CoDing Sequences) [9] by launching a BLAST [1] program and then construct the alignment using ClustalW [3] program. Similar tools using backtranslation to perform an alignment are available in [2] and [14]. Additionally, backtranslation can be made for direct derived nucleic sequences search from a protein sequence. This search is based on IUB/IUPAC degenerate nucleotide base codes [13] and it assumes the use of a consensus sequence to represent all possible sequences corresponding to the original protein. With this approach, one protein is backtranslated to a unique IUB codon that represents all possibilities in only one nucleotide triplet. Software that use IUB codes to backtranslate a protein only uses a replacement of amino acids by corresponding IUPAC codons. Consequently, to obtain a nucleic sequence from an input peptide, we need to produce all the potential codons.

Reverse translation can be also used to compute probable DNA underlying the reported amino acid substitution by selecting reference amino acid in regards to the codon usage preference for each organism [8]. The calculation is based on user-selected codon table. Several reverse translation tools are based on probability computation or statistical determination of amino acid codon referred to the nucleic databanks. It can use genetic algorithms [5], [6] to determine the most probable codon at a given position by launching a data mining algorithm on a selected databank. In addition, some software use neural networks data mining to similarly perform the same task **[**16**].** Another method based on hierarchical clustering data mining to do backtranslation was proposed in [4]. Several methods use particularly data mining algorithms to find the most adapted reverse translation of a peptide. For example, Serine (S) amino acid can be encoded by 6 triplet nucleic codons: TCA, TCC, TCG, TCT, AGC and AGT. According to databanks, the probability to find a peptide with Serine amino acid which arises from a translation of nucleic sequence and having AGC codon is 26% when *E. coli* species databank is used as reference databank. So, when S amino acid is backtranslated, the tool presented in [13] returns the most probable codon based on species characteristics. Furthermore, similar tools use the most probable codon or user-selected codons to perform such a reverse translation and are detailed in [10], [17], [18] and [19]. Reverse translation based on distance matrix providing codon usage between given sequences, is shown in [4]. Backtranslation programs are often associated with complete software or packages as [8] and [13], thus belonging to a bigger application, which uses other elementary programs to perform bioinformatics tasks.

Whatever the approach, the backtranslation problem presents an actual complicatedness due to the genetic code degeneracy. To have all possible non degenerate DNA sequences corresponding to an input peptide, to the best of our knowledge, no tool is available to automatically perform this task. However, working on oligopeptides with a limited length becomes reasonable with the current storage capacities. The problem to tackle is of exponential nature, but it is still tractable with an interesting size of oligopeptide which will produce less than 3 terabytes (6^{16}, knowing that we can store a codon on a single byte). Of course if the degeneracy is favourable we can backtranslate much longer oligopeptides.

In this study, we introduce a new algorithm called *DegenRev* that performs a high performance backtranslation of oligopeptides despite. The algorithm is based on a learning step which previously encodes the degeneracy for each step of the algorithm.

2 *DegenRev*: the algorithmic model

The codon table data used in *DegenRev* is stored in an appropriate structure that facilitates amino acid accession for each processed peptide. All codons are stored in an array of string. But the most important part of our algorithm is its approach which takes advantages of the degeneracy of each amino acid to calculate remaining oligos at a given step of peptide processing. First, an input peptide is stored in an array. For each amino acid *i* we calculate its *cumulated degeneracy d(i)*, the *remaining degeneracy r(i)* and the *passed degeneracy c(i)*. We note *NC(i)* the number of codons of the amino acid *i; i* ∈ [0, *pl*-1]. So, the total peptide degeneracy having length *pl* is given by the equation (1):

$$D = \prod_{i=0}^{pl-1} NC(i) \qquad (\mathbf{1})$$

The cumulated degeneracy to the i^{th} amino acid is given by the equation (2):

$$d(i) = \prod_{k=0}^{i} NC(k) \qquad (\mathbf{2})$$

The remaining degeneracy at the i^{th} amino acid is given by the equation (3):

$$r(i) = \prod_{k=i+1}^{pl} NC(k) \qquad (\mathbf{3})$$

And the passed degeneracy at the i^{th} amino acid is given by the following formula:

If (*i = 0*) **then**
c(i) = 1

Else

$$c(i) = \prod_{k=1}^{i} NC(k-1)\,. \qquad (\mathbf{4})$$

While the initialization of all variables is done, the translation step begins. The program tests for each amino acid if *r(i)* is reached. If it is the case, then the next amino acid is processed, else the current one is used. This process will be repeated *c(i)* times as it is described in figure 1. *DegenRev* performs exactly *D* loops using this approach without loosing time on reinitialising degeneracy variables. For an amino acid *i,* the degeneracy is given by equation (5):

$$D = d(i) \times r(i) \qquad (\mathbf{5})$$

The obtained oligos (nucleic sequences generated after reverse translation of the peptide) are ordered by their respective degeneracy. Figure 1 shows an example:

Output results for the sample oligopeptide '*KIL*' are done by the following variables:

OligoPeptide:	***K***	***I***	***L***	
i:		**0**	**1**	**2**
NC(i):	**2**	**3**	**6**	
D:		**36**		
d(i):		**2**	**6**	**36**
r(i):		**18**	**6**	**1**
c(i):		**1**	**2**	**6**

As shown here, our technique takes advantage of the degeneracy pre-storage to perform backtranslation task. The algorithm writes data in an output file containing non redundant reverse translation possibilities. Each computed oligo will represent a potential coding sequence part for the studied protein. The complexity of the *DegenRev* algorithm is given by the following equation:

$$Complexity = D \times pl \qquad (\mathbf{6})$$

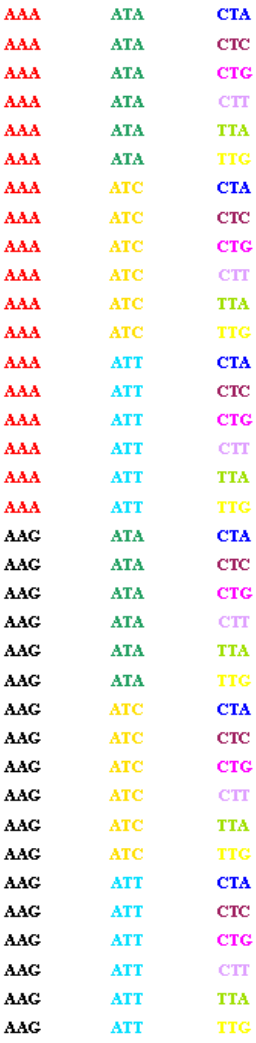

Fig. 1. The oligos generated from the peptide "KIL" and ordered by the algorithm. Each amino acid's repetition depends on passed degeneracy *c (i)* done in formula (4) which gives exactly the number of current remaining computed codon. Colours are only used to differentiate codons. Oligos should be taken line by line which is the order of calculation used by the algorithm.

The following is a pseudo-code of the reverse translation algorithm *DegenRev*. The main loop, in which backtranslation is performed using previously described degeneracies supposed to be known for the following sub-program, is described. In each loop, *DegenRev* build current generated oligo using its own degeneracy as shown in Figure 1. The order depends on the initialization structure made in the beginning of the algorithm.

```
Function DegenRev
Input:
                Peptide pep
Output:
                File oligos.out
Variables:
                Array of integers: k, l, j
                Array of characters: oligo
Begin
  While m < D do
    for i = 0 to pl do
      Concatenate (oligo,codon_l[i](pep[i]));
      if (k[i] = r[i] - 1) then
        k[i] := 0;
        if (l[i] = codon_number(pep[i]) - 1) then
          l[i] := 0;
          if (j[i] = c[i]-1) then
            j[i] := 0;
```

```
          else
            j[i]++;
          end if
        else
          l[i]++;
        end if
      else
        k[i]++;
      end if
    end for
    write(oligo, oligos.out);
    m := m + 1;
    oligo := ∅;
  end while
End
```

3 *DegenRev* Performances

The algorithm was tested on a machine with a Xeon processor at 2.40-GHz and 2-GB RAM. As mentioned in the previous section, the program was developed in the C language and was compiled with the 3.4.1 version of GCC compiler with 64 bits system options to allow LFS (Large File Support). A first test of our algorithm was performed based on incremented degeneracy and minimum length i.e. for a given degeneracy of input peptide the minimum length was taken. For example, let be d = 373248. This degeneracy is obtained by several peptides having different lengths: IAARRRRR (8), KKKRRRRRR (9), KIIIIAAARR (10), and KKKKIRRRRR (11) etc... However, IAARRRRR was selected for this degeneracy because it is the smallest peptide that has degeneracy d.

3.1 General test statistics

In this section, the general behaviour of our algorithm is described, particularly when it is used to calculate all possible oligos for a given input peptide. Additionally, two aspects of our algorithms can be distinguished depending on the writing step. Indeed, first, the algorithm with the writing phase was tested in order to have the real time consumption and, then, it was tested without this phase in order to obtain its performance for very high degeneracies because the disk space limit can be reached easily when high degeneracy peptides are computed. In addition, when the algorithm is applied to real data, the writing step will concern only the part of oligos that have the best specificity and sensitivity characteristics for probe design. Execution time was given by the *clock ()* function of C library and it returns the CPU time of our program.

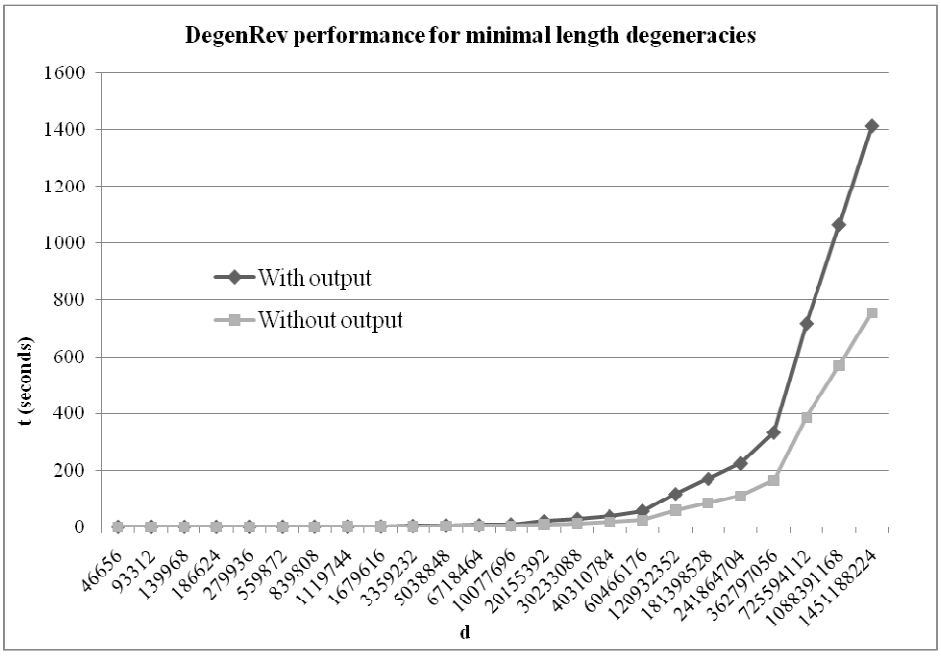

Fig. 2. The global performance (t which consists of the execution time in seconds) of DegenRev algorithm using minimum length peptide degeneracy d. The given data concerns DegenRev with output file and without output file. The two curves have the same exponential look. The difference between the two curves is due to writing oligos results into an output file.

In figure 2, the general performance of *DegenRev* algorithm shows that the algorithm is extremely efficient for short peptide having degeneracy about 10^7. In fact, to have all possible oligos for short peptides, *DegenRev* takes less than 1 minute on current desktop computers. The execution time increases significantly when degeneracy is greater than 10^8. Additionally, to avoid disk quota limitation, the size of output file for each peptide should be calculated and compared to the available free disk space. Table 1 and figure 2 show that the execution time depends both of peptide length and its degeneracy.

Table 1. For each tested peptide, the total degeneracy and the size of the output file increases when the degeneracy rises.

Peptide	Degeneracy	Output file size (in Bytes)
RRRRRR	46656	886464
RRRRRRK	93312	2052864
RRRRRRI	139968	3079296
RRRRRRA	186624	4105728
RRRRRRR	279936	6158592
RRRRRRRK	559872	13996800
RRRRRRRI	839808	20995200
RRRRRRRA	1119744	27993600
RRRRRRRR	1679616	41990400
RRRRRRRRK	3359232	94058496
RRRRRRRRI	5038848	141087744
RRRRRRRRA	6718464	188116992
RRRRRRRRR	10077696	282175488
RRRRRRRRRK	20155392	624817152
RRRRRRRRRI	30233088	937225728
RRRRRRRRRA	40310784	1249634304
RRRRRRRRRR	60466176	1874451456
RRRRRRRRRRK	120932352	4111699968
RRRRRRRRRRI	181398528	6167549952
RRRRRRRRRRA	241864704	8223399936
RRRRRRRRRRR	362797056	12335099904
RRRRRRRRRRRK	725594112	26846982144
RRRRRRRRRRRI	1088391168	40270473216
RRRRRRRRRRRA	1451188224	53693964288

In the following section we will describe the behaviour of our algorithm when oligopeptide length varies.

3.2 Length influence on *DegenRev* performance

According to figure 3, the program conserves a linear behaviour when the input peptide length increases and the degeneracy is keep constant.

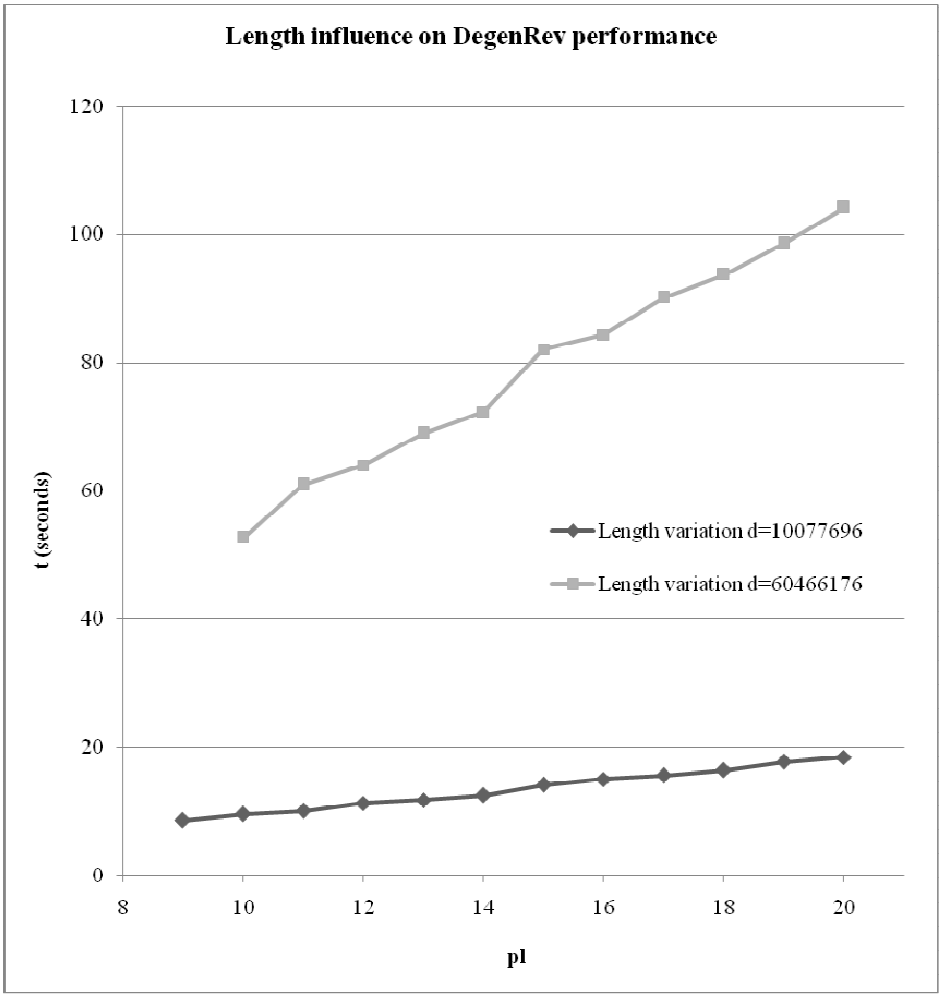

Fig. 3. *pl* is the input peptide length used in equation (1) to compute the total degeneracy which is constant in this figure. Linear like curves which represent *DegenRev* behavior when the degeneracy is fixed at 10077696 and at 60466176 and when the length is varying. *t* is the execution time in seconds.

To confirm these results we tested our algorithm for a degeneracy d = 60466176 and we obtained the results described in figure 3. Our algorithm behaves as a linear function when the degeneracy is fixed. Moreover, The curve slope increases when degenerecies are greater than d as attested by the curve of degeneracy 60466176, which has a slope equal to about 5 while the curve of degeneracy 10077696 has only a slope equal to about 1. *DegenRev* can compute efficiently oligos for peptides having a small degeneracy. The main aim is to have a degeneracy range that minimizes the whole execution time for a group of peptides representative of enzymatic characteristics. This program will be used as a part of more consequent software used to design oligonucltides for microarrays. *DegenRev* allows the computation of a large choice of oligopeptide lengths for oligos that will serve as input for enzyme specific oligos design.

3.3 Degeneracy influence on *DegenRev* performance

This section shows *DegenRev* behavior when input peptide length is fixed allowing us to make a decision about optimal length and degeneracy combination range.

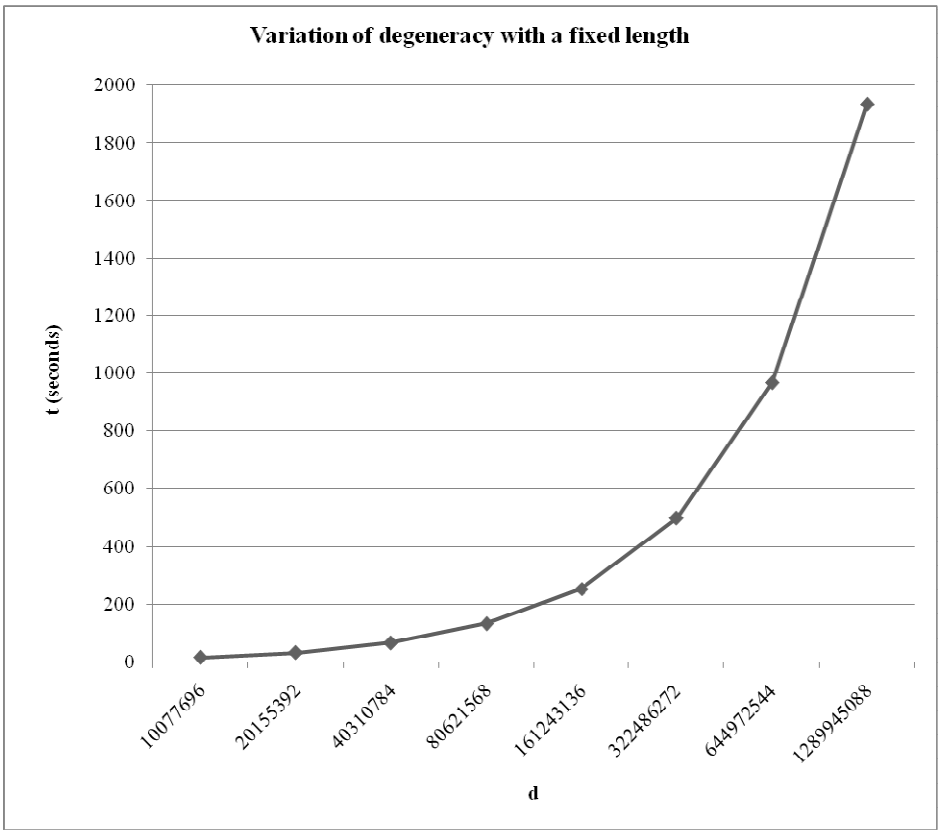

Fig. 4. *pl* = 18.*DegenRev* behavior when the length is fixed and degeneracy is varying. *t* is the execution time in seconds and *d* is the degeneracy.

Obviously and as expected, the input peptide degeneracy is the most influent factor on the execution time. In fact, the variation of degeneracy with a fixed length shows a more important increase of the execution time than the variation of the oligopeptides length with a fixed degeneracy.

3.4 *DegenRev* tested on real datasets

In a bioremediation context, we tested the algorithm on data collected from the bacteria *Pseudomonas* sp LZT5 which has a gene that encodes naphthalene dioxygenase. We took the coding sequence (CDS), which represents the enzyme involved on those degrading capabilities, and that are referred in the UniProt/TrEMBL databank by the accession number "Q3LTH2". Then, we extract all peptides having a length of 11 amino acids. This size limit corresponds to 33 mers oligos which is an interesting size for the design of short oligonucleotides [12]. Therefore, we obtained 175 oligopeptides ready to be tested by our algorithm to show all oligos derived from this enzyme. The test consists of creating a file containing all those oligopeptides; one peptide per line. Then full backtranslation is computed for each oligopeptide. Execution time for all the data is finally collected. *DegenRev* without file output was tested because we plan to apply our oligo design program [11] at run time for each found oligo. The results in figure 5 show that the whole execution time is of 101 seconds. This is relatively short for 175 oligopeptides of 11 amino acids compared to the simulated data of table 1. The chosen length represents a significant scale for probe design because it generates oligos having a length of 33, which is the top limit for short oligos. According to such results, we see that our algorithm shows a real capability of computing full backtranslation on real datasets with a very reasonable execution time. Moreover, according to the results shown in figure 3, we know that a wide range of oligopeptides length is able to be used by our efficient full backtranslation program.

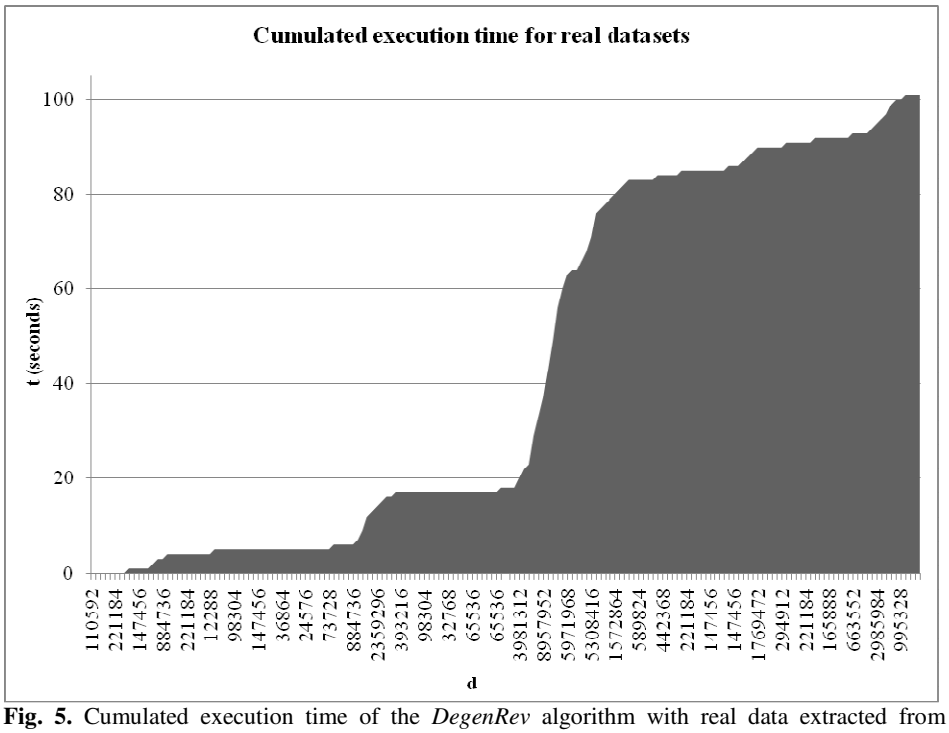

Fig. 5. Cumulated execution time of the *DegenRev* algorithm with real data extracted from *Pseudomonas* sp LZT5 data. Each graduation corresponds to an oligopeptide of the 175 found. Visible degeneracies are automatically and randomly selected ones among full data. *d* is the degeneracy and *t* is the execution time.

4 Conclusion and discussions

We presented in this paper a new algorithm, *DegenRev* and an approach using full backtranslation of oligopeptides for the design of microarray oligonucleotides. Existing algorithms provide solution with IUB degenerated code. The algorithm introduced is efficient on simulated and real data, and it is the only one to give all the possible oligos coding for a given oligopeptide. *DegenRev* is very appropriate for peptide having degeneracy smaller than 10^9 independently from the peptide length, which influences on execution time very slowly. The main limitation of our algorithm is the disk space issue, since a few terabytes can be needed for large real datasets. However, the new hard disks for personal computers in manufacturers research laboratories are above 5 terabytes, they will soon be available on our desktops. In addition, in the near future, we will study the parallelization of this algorithm on computing clusters and grids.

Acknowledgments. This work was financially supported by the regional council of Auvergne (France) in the INSTRUIRE project. The authors thank Dr. Christophe Gouinaud for expert technical assistance.

References

1. Altschul, S., Gish, W., Miller, W., Myers, E.W., Lipman, D.J.: Basic local alignment search tool. Journal of molecular biology 215 (1990) 403-410.
2. Bininda-Emonds, O.R.: transAlign: using amino acids to facilitate the multiple alignment of protein-coding DNA sequences. BMC Bioinformatics 6 (2005) 156.
3. Chenna, R., Sugawara, H., Koike, T., Lopez, R., Gibson, T.J., Higgins, D.G., Thompson, J.D.: Multiple sequence alignment with the Clustal series of programs. Nucleic Acids Research 31 (2003) 3497-3500.
4. Ma, J., Zhou, T., Gu, W., Sun, X., Lu, Z.: Cluster analysis of the codon use frequency of MHC genes from different species. Biosystems 65 (2002) 199-207.
5. Moreira, A.: Genetic algorithms for the imitation of genomic styles in protein backtranslation. Theoretical Computer Science 322 (2004) 297-312.
6. Moreira, A., Maass, A.: TIP: protein backtranslation aided by genetic algorithms. Bioinformatics 20 (2004) 2148-2149.
7. Moretti, S., Reinier, F., Poirot, O., Armougom, F., Audic, S., Keduas, V., Notredame, C.: PROTOGENE: turning amino acid alignments into bona fide CDS nucleotide alignments. Nucleic Acids Research 34 (2006) W600-W603.
8. Nash, J.: A computer program to calculate and design oligonucleotide primers from amino acid sequences. Computer applications in the biosciences 9 (1993) 469-471.
9. Nirenberg, M.W.: Historical review: Deciphering the genetic code--a personal account. Trends in biochemical sciences 29 (2004) 46-54.
10. Pesole, G., Attimonelli, M., Liuni, S.: A backtranslation method based on codon usage strategy. Nucleic Acids Research 16 (1988) 1715-1728.
11. Rimour, S., Hill, D., Militon, C., Peyret,P.: GoArrays: highly dynamic and efficient microarray probe design. Bioinformatics 21 (2005) 1094-1103.

12. Relogio, A., Schwager, C., Richter, A., Ansorge, W., Valcarcel, J.: Optimization of oligonucleotide-based DNA microarrays. Nucleic Acids Research 30 (2002) e51.
13. Stothard, P.: The sequence manipulation suite: JavaScript programs for analyzing and formatting protein and DNA sequences. Biotechniques 28 (2000) 1102, 1104
14. Suyama, M., Torrents, D., Bork, P.: PAL2NAL: robust conversion of protein sequence alignments into the corresponding codon alignments. Nucleic Acids Research 34 (2006) W609-W612
15. Wernersson, R., Pedersen, A.G.: RevTrans: multiple alignment of coding DNA from aligned amino acid sequences. Nucleic Acids Research 31 (2003) 3537-3539
16. White, G., Seffens, W.: Using a neural network to backtranslate amino acid sequences. Electronic Journal of Biotechnology 1 (1998) 17-18
17. WebLab backtranseq: http://weblab.cbi.pku.edu.cn/program.inputForm.
18. CLC Protein Workbench: http://www.clcbio.com/index.php?id=93
29. Gene Design - Reverse translation: http://slam.bs.jhmi.edu/cgi-bin/gd/gdRevTrans.cgi