PRT: Parallel program for a full backtranslation of oligopeptides

Mohieddine Missaoui, Cécile Militon, David R.C. Hill, Christophe Gouinaud, Pierre Peyret

PRT: Parallel program for a full backtranslation of oligopeptides

Mohieddine MISSAOUI[1,2], Cécile MILITON, David HILL, Christophe GOUINAUD, and Pierre PEYRET

Research Report LIMOS/RR-O7-11

07 juin 2007

[1]LBP – UMR CNRS 6023 Complexe des Cézeaux
[2]LIMOS – UMR CNRS 6158 Complexe des Cézeaux

missaoui@isima.fr

ABSTRACT

DNA hybridization methods have become the most widely used tools in molecular biology to identify organisms and evaluate gene expression levels. PCR (Polymerase Chain Reaction)-based methods, fluorescent in situ hybridization (FISH) and the recent development of DNA microarrays as a high throughput technology need efficient primers or probes design. Evaluation of the metabolic capacities of complex microbial communities found in terrestrial or aquatic environments requires new probe design algorithms that reflect the genetic diversity. As only a small part of the microbial diversity is known, gene sequences deposited in international databases do not reflect the entire diversity. In this context we propose to use oligopeptide sequences for the design of complete set of DNA probes that are able to target the entire genetic diversity of genes encoding enzymes. Due to the degenerated genetic code backtranslation must be managed efficiently. To our knowledge no software has been developed to propose a full backtranslation. This complexity is tractable since we only need to focus on short oligopeptides for DNA probe design. We propose new algorithms that perform a high performance oligopeptide backtranslation into all potential nucleic sequences. We use different efficient techniques such as memory mapping to perform such a computing. We also propose a MPI parallel computing that reduces the whole execution time using data load balancing and network file stream distribution on a cluster architecture.

Index Terms-- **Oligopeptide, Probe design, Full back translation, Parallel computing,**

I. INTRODUCTION

The overwhelming majority of life on our planet is microbial, both in terms of phylogenetic diversity and sheer number of organisms. Virtually every conceivable environmental niche harbours microorganisms capable of growing there. Traditional microbiological methods of cultivation recover less than 1% of the total bacterial species, and the cultured portion of bacteria is not representative of the total phylogenetic diversity. Considering the extent of functional diversity described for microbes and the numerous applications of their secondary metabolites, the biotechnological potential hidden among the 99% of the bacteria that are un-cultured is immense. Deciphering the microbial metabolic pathway capacities in complex environments (soils, lakes, oceans…) is essential for the protection of the environment. However, conventional biochemical and molecular methods (PCR-based technologies) for assessing microbial community structure and activities are labour-intensive. Microarrays are a powerful tool for the parallel, high throughput detection and quantification of many nucleic acid molecules [3]. Oligonucleotide microarrays are the most widely developed tools for microbial diagnostics. However, probe design is not a trivial task and needs particular attention [15]. Functional gene arrays (FGAs) contain probes corresponding to genes encoding key enzymes involved in various ecological and environmental processes such as carbon fixation, nitrification, denitrification, sulphate reduction, methane oxidation, and contaminant degradation. Both PCR-amplified DNA fragments and oligonucleotides derived from functional genes can be used to manufacture FGAs. In each situation we need to know the gene sequences for the probe design. As a large part of microbial diversity is unknown we can only use sequences from genes deposited in international databases. To develop an explorative aspect for functional microarrays we propose to design probes derived from the protein sequences. In this context we are able to determine the probes sets that characterize an enzyme or a metabolic pathway of interest. We have developed full backtranslation algorithms for oligopeptides able to provide interesting DNAs sequences for probe design in the context of functional microarrays.

II. Materials and Methods

A. *Related Works*

Parallel computing is often the best solution to optimize bioinformatics algorithms which are time and memory consuming [17]. Computer architectures such as computing clusters and grids are now appropriate for many bioinformatics applications [9]. The backtranslation problem appeared after the protein synthesis mechanism discovery [13] followed by the entire genetic code decoding in 1961 by the Nobel Prize laureate M. Nirenberg. Therefore, Bioinformatics was incidental to biology advancement and contributed to solving difficult problems such as local and global sequences alignment. Backtranslation is most often required in larger applications using protein to DNA or RNA passing. Studying phylogeny uses genetic code to perform a phylogenetic tree construction based on sequence homology detection. Information detained by proteins is useful when constructing optimal multiple DNA alignments for phylogenetic analysis [20]. In this case, backtranslation is simply a replacement of amino acid by the existing codon in the initial corresponding nucleic sequence. Thus, passing from protein to DNA can preserve this phylogenetic information.

Nevertheless, genomic and transcriptomic databases are used as a source of sequences existing in protein alignment, so it is possible to find CDS (CoDing Sequences) [11] by launching a BLAST [1] program and then construct the alignment using the ClustalW [6] program. Similar tools that use backtranslation to perform an alignment are available [2] [17]. Moreover, using backtranslation can be made for direct derived nucleic sequences search from a protein sequence. Most of them are based on IUB/IUPAC degenerate nucleotide base codes [14], which assume the use of a consensus sequence to represent all possible sequences corresponding to the initial protein. Figure 1 shows consensus triplet codons of all amino acids using IUB degenerate nucleotide base codes. With this approach, one protein is backtranslated to a unique IUB codon that represents all possibilities in only one nucleotide triplet. Software that uses IUB codes to reverse translate a protein only uses a replacement of their amino acids by corresponding codons. Consequently, to obtain a nucleic sequence from input peptide, all we have to do is to combine the known codons in the appropriate order.

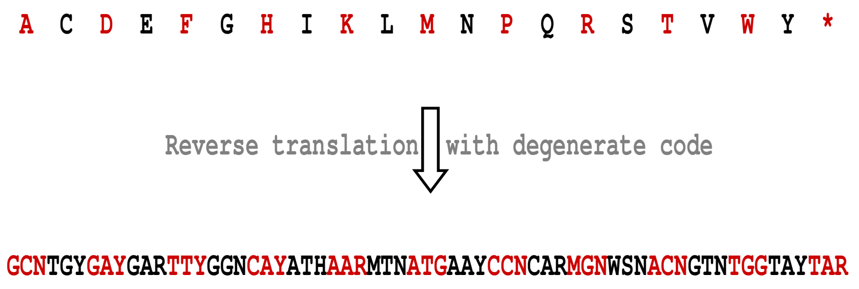
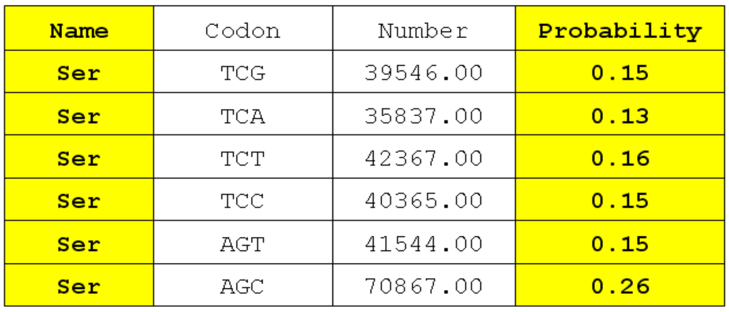

A C D E F G H I K L M N P Q R S T V W Y *
Reverse translation with degenerate code
GCNTGYGAYGARTTYGGNCAYATHAARMTNATGAAYCCNCARMGNWSNACNGTNTGGTAYTAR

Fig. 1. Consensus nucleic sequence of a backtranslated peptide using IUB degenerate nucleotide base codes. This example shows all possible reverse translations (20 amino acids plus “STOP” codons given by ‘*’).

Reverse translation can be also used to calculate probable DNA codon usage for an organism [12]. It can use genetic algorithm [10] to determine the most probable codon at a given position by launching a data mining algorithm on a selected databank. In addition, some software uses neural networks data mining to similarly perform the same task [21]. An additional method is based on hierarchical clustering data mining to do this backtranslation [8]. We note that several methods use particularly data mining algorithms or distance matrix to find the most adapted reverse translation of a peptide [16]. The software returns the most probable codon for a given species as shown in Figure 2. Similar tools which use the most probable codon or user-selected codons to perform such a reverse translation are available by web interface [21, 22, 23].



Name	Codon	Number	Probability
Ser	TCG	39546.00	0.15
Ser	TCA	35837.00	0.13
Ser	TCT	42367.00	0.16
Ser	TCC	40365.00	0.15
Ser	AGT	41544.00	0.15
Ser	AGC	70867.00	0.26

Fig. 2 Nucleic acid sequence determination by backtranslation based on the preferential genetic code usage in E. coli species. (See http://cgpdb.ucdavis.edu/database/sms/javascript/index.html for more details).

From the previous works, and to the best of our knowledge no tool is available to automatically perform a full backtranslation providing all possible non degenerate DNA sequences corresponding to an input oligopeptide. In the next section we present high performance parallel algorithms able to use cluster and grid resources to take up this challenge. This work is limited to oligopeptide since it is practically impossible to produce all the DNA sequences corresponding to a protein, or even to a long peptide. The main reason is simply the unfeasibility of storing the results because of space and time constraints.

B. Why do we need a complete backtranslation?

Our method aims to show all potential DNA sequences that could spark a given oligopeptide sequence for the development of functional microarrays used in microbial ecology. Our approach supposes that the maximum length of the DNA generated sequences does not exceed 24 bases because the target sequences are probes which will serve in biochip design such as what we proposed in [15]. Thus, the given peptides used on backtranslation process will be about 6 to 8 amino acids. Furthermore, backtranslation is still ambiguous due to its memory-time consuming nature mainly caused by the genetic code degeneracy. Two approaches are presented here; both use the genetic code degeneracy to construct the algorithm. The main difference between the two approaches relies on data manipulation.

C. *The Mathematical Model*

We propose a new algorithmic approach to construct backtranslated DNA sequences from a protein sequence. Let $\Gamma = \{A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y, *\}$ and $\Sigma = \{A, C, G, T\}$ be the alphabets for amino acids and nucleotides, respectively. Let $\varphi : \Sigma^3 \rightarrow \Gamma$ be the translation of a DNA sequence according to the genetic code. We denote $card(\varphi^{-1}(X))$ the number of triplet codons that code for the amino acid '*X*'. It is called the degeneracy of the amino acid '*X*'. We named the algorithm PRT for oligo**P**eptide **R**everse **T**ranslation. This algorithm can be assimilated to a function:

$$PRT_n\,(p): \Gamma^n \rightarrow \delta_n(p) \times (\Sigma)^{3 \times n} \qquad (1)$$

Given an oligopeptide 'p' of length 'n', the function will return $PRT_n(p)$ DNA sequences; $PRT_n(p)$ is the total degeneracy of the input sequence 'p'; it depends both on the sequence and on its length. Therefore, for an oligopeptide $X_1X_2X_3...X_n$, we have:

$$\delta_n\,(p) = \prod_{i=1}^{n} \mathrm{card}\left(\varphi^{-1}(x_i)\right) \qquad (2)$$

Our algorithm uses the PRT_n procedure (see equation 1) to compute non degenerated DNA sequences.

D. *Algorithm principle*

Let $p = X_1X_2X_3...X_n$ be a peptide of *'n'* amino acids. Let X_i ; $1 < i < n$ be an amino acid of the peptide *'p'* in position *'i'* (see figure 3). We denote $d(i)$ the calculated degeneracy from the initial position to position '*i*'.

$$d(i) = \prod_{k=1}^{i} \mathrm{card}\left(\varphi^{-1}(x_k)\right) \; (3)$$

We denote *r(i)* the remaining degeneracy in step '*i*':

$$r(i) = \frac{\delta_n(p)}{d(i)} \quad (4)$$

We describe in details our algorithm as follows:

(1) For the *i*th amino acid, append current codons according to degeneracy variable (see below) in the appropriate order (see the following algorithm)

(2) Save the result in a structure

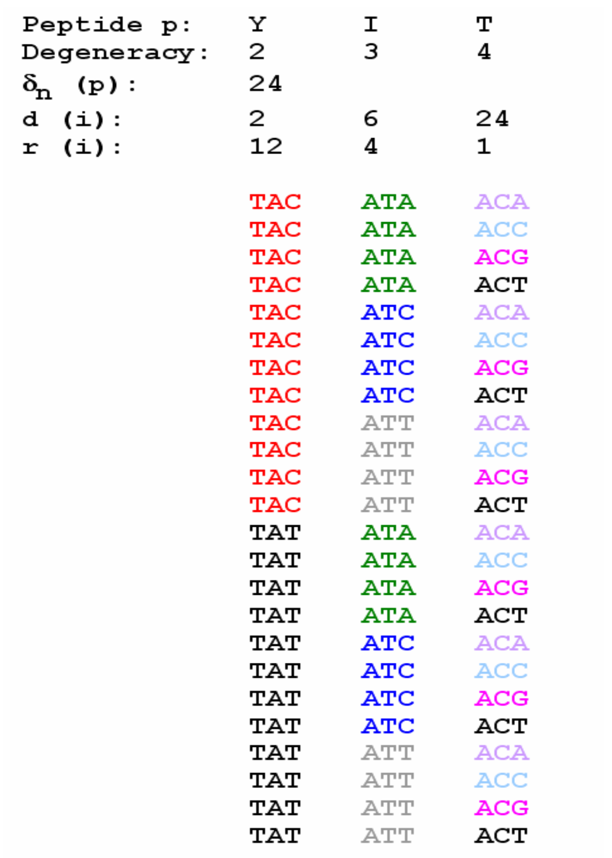
(3) Re-compute current degeneracy variables

(4) Reinitialize counters

Algorithm: procedure

```
Input: peptide of n length

Output: file of generated oligos

while i <= n do

repeat (for each codon of $x_i$)

d (i)/card ($\varphi^{-1}$($x_i$)) times

   repeat append (codon) r (i) times;

end repeat

    end repeat

    i = i + 1;

end while
```

This algorithm can be transposed using several techniques. We implemented the different techniques using the C language capabilities. In this paper we show the most efficient strategies we obtained. Additionally, our approach depends neither on the peptide length nor on the sequence composition. The latter are computed by the algorithm according to the codons table.



Peptide p:	Y	I	T
Degeneracy:	2	3	4
δ_n (p):	24		
d (i):	2	6	24
r (i):	12	4	1
	TAC	ATA	ACA
	TAC	ATA	ACC
	TAC	ATA	ACG
	TAC	ATA	ACT
	TAC	ATC	ACA
	TAC	ATC	ACC
	TAC	ATC	ACG
	TAC	ATC	ACT
	TAC	ATT	ACA
	TAC	ATT	ACC
	TAC	ATT	ACG
	TAC	ATT	ACT
	TAT	ATA	ACA
	TAT	ATA	ACC
	TAT	ATA	ACG
	TAT	ATA	ACT
	TAT	ATC	ACA
	TAT	ATC	ACC
	TAT	ATC	ACG
	TAT	ATC	ACT
	TAT	ATT	ACA
	TAT	ATT	ACC
	TAT	ATT	ACG
	TAT	ATT	ACT

Fig. 3. An example to highlight the algorithm operation for backtranslation

E. *Computer systems*

The computer used for this study is a bi-processor equipped with an Intel$^{(R)}$ Xeon^{TM} at 2.4 Ghz with 2 Gb of RAM. For sequential tests, we used a single processor. We developed the algorithm using the C language with a *GNU C* compiler (version 3.4.1).

F. *Backtranslation strategies*

Two strategies based on the PRT$_n$ algorithm are presented here. A comparison between the two implementations will give us decisive elements in order to retain the best solution for a specific problem.

1) *Dynamic memory allocation*

The first strategy adopted to develop the PRT algorithm was to use a large array pre-allocated in dynamic memory. We used a pointer data structure to store output data consisting of generated backtranslated DNA sequences from the input peptide. Figure 4 shows the structure used to save the generated nucleic acid sequences. The computing algorithm is used with a dynamic memory allocation strategy in order to have all sequence possibilities. The advantage of this strategy is to have access to DNA sequences at any time and the computing allows storage of the output in a file.

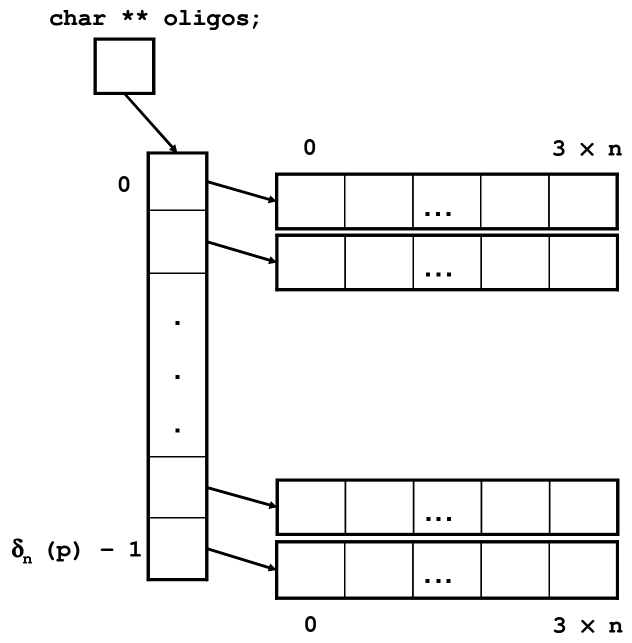


Fig. 4. Dynamic Allocation main storage data structure. This C data structure is used to store all generated probes

Furthermore, the algorithm uses this data structure and the codon table structure. A single line of code is required to store data in the structure. The storage of sequences follows the data structure presented in figure 4 exactly.

2) *File memory mapping*

The second method used to perform backtranslation is based on file memory mapping. We launch the algorithm to write the output nucleic sequences in a memory file managed like a pointer. In our C code, we use appropriate structures such as start and end mapped file pointers. First of all, we create a file formatted according to the output. Then, each codon is placed by the algorithm in the appropriate position in the file. The position is computed for each amino acid. Using file memory mapping management strategy is particularly interesting because in opposition to dynamic memory allocation we save sequences directly in the output file and we don't introduce an intermediate data-structure. Though we rely on a file, we are limited by the system RAM as with the dynamic allocation method. Copying results (codons) is done nucleotide by nucleotide for each codon. This slows down the execution time, but it remains more efficient than the dynamic memory allocation strategy. We copy codons to the output oligos file using a file descriptor pointer that moves by finding the appropriate position of each codon. The output file is used like a table structure so we can easily manipulate its indexes.

The method can be used with memory mapping only if we assume that output oligos are stored in a file because we made preliminary actions and dedicated variables initialization to run this algorithm : we should declare a size, a file status and a file descriptor to be able to use memory mapping capabilities. Our algorithm uses different pointers to move along the mapped file.

III. RESULTS

A. *Sequential Performance comparison*

Execution time decreases when we use the memory mapping approach because we write the nucleic sequences directly in the output file, as shown in figure 5. We launched the tests on an isolated computer to assert that the system overhead and load was the same.

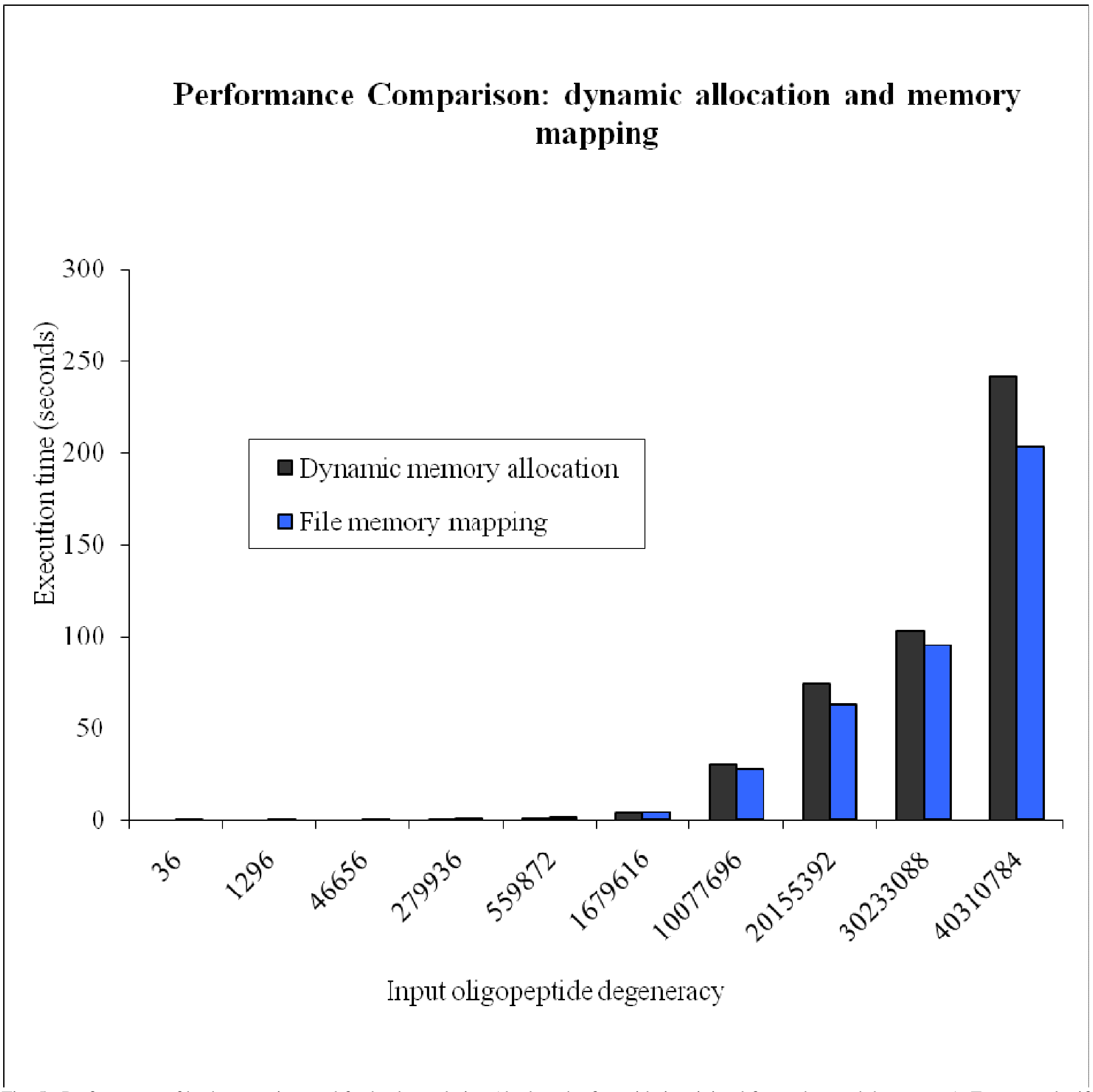
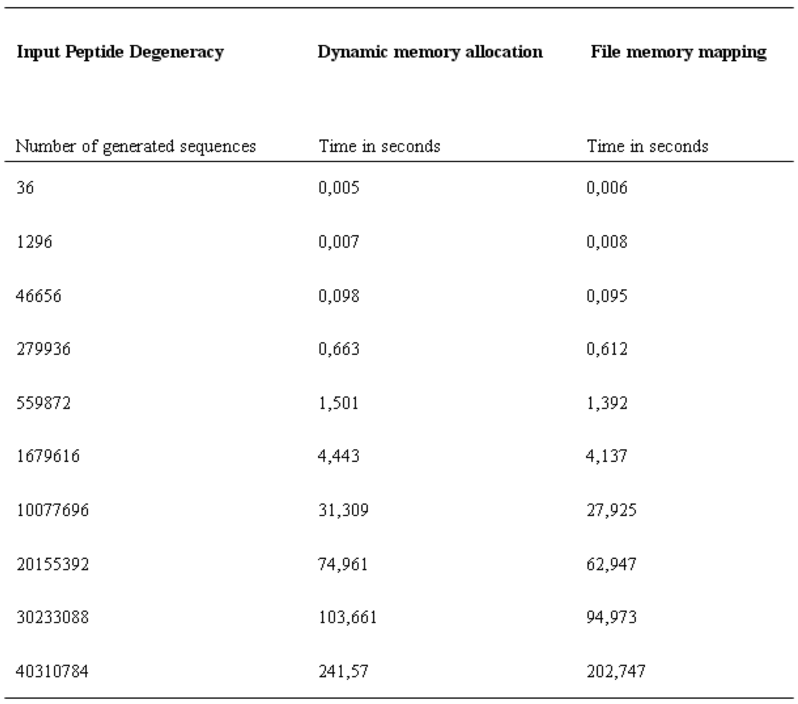


Fig. 5. Performance of both strategies used for backtranslation (the length of peptide is minimal for each tested degeneracy). For example, if we take degeneracy 20155392 it corresponds to RRRRRRRRRK oligopeptide which is the shortest peptide for this degeneracy.

In figure 5, the difference in execution time between the two strategies is due to the memory allocation performed by the dynamic approach. When the dynamic structure is full we print out the results in an output file of oligonucleotides. To be able to compare the two strategies we must have an unlimited memory system; which is not reasonable. Table 1 shows execution time of the two backtranslation strategies. We reported the whole execution time returned by Linux '*time*' function.

TABLE 1. DATA OF PERFORMANCE COMPARISON FOR THE TWO STRATEGIES USED FOR BACKTRANSLATION. IT BELONGS TO FIGURE 5.



Input Peptide Degeneracy	**Dynamic memory allocation**	**File memory mapping**
Number of generated sequences	Time in seconds	Time in seconds
36	0,005	0,006
1296	0,007	0,008
46656	0,098	0,095
279936	0,663	0,612
559872	1,501	1,392
1679616	4,443	4,137
10077696	31,309	27,925
20155392	74,961	62,947
30233088	103,661	94,973
40310784	241,57	202,747

As expected, we observe that if the peptide degeneracy increases, the performance exponentially decreases. In fact, our algorithm has to handle an exponential complexity, and even if the problem is tractable for oligopeptides, it can require a significant amount of disk space. . This disk space (RDS : Required Disk Space) can easily be predicted for a given oligopeptide. It depends on **(p)** and is given by the following equation:

$$\mathrm{RDS} = \left[\delta_n(p) \times (n \times 3 + 1)\right] \mathrm{Bytes} \qquad (5)$$

For an oligopeptide having a degeneracy around 10.10^6, a single computer (see the Computer System section) requires less than 300 Mb of disk space and less than 30 seconds of execution time in the case of memory mapping strategy. In the study described by Tamura et al., (1991), the degenerate code used to a find consensus nucleic sequence backtranslated from a peptide does not require more than 3 times the size of the input peptide. In our new approach, we should save all obtained sequences for only one oligopeptide. In practice, we need to work with several thousand of oligopeptides. However, there is no need to keep the resulting DNA sequences for all oligopeptides, they can be processed to design DNA probes one by one and peptide by peptide. However, in order to increase the throughput, parallel computing is well-adapted to reduce the complexity of this algorithm. In fact, the backtranslation step is time-consuming because we try to backtranslate a large number of oligopeptides knowing that the whole execution time increases when we work on long oligopeptides. In the next section, we present Parallelization methods for our two algorithm variants using the MPI library. The algorithms developed were run on a local cluster architecture.

B. *Parallel implementation*

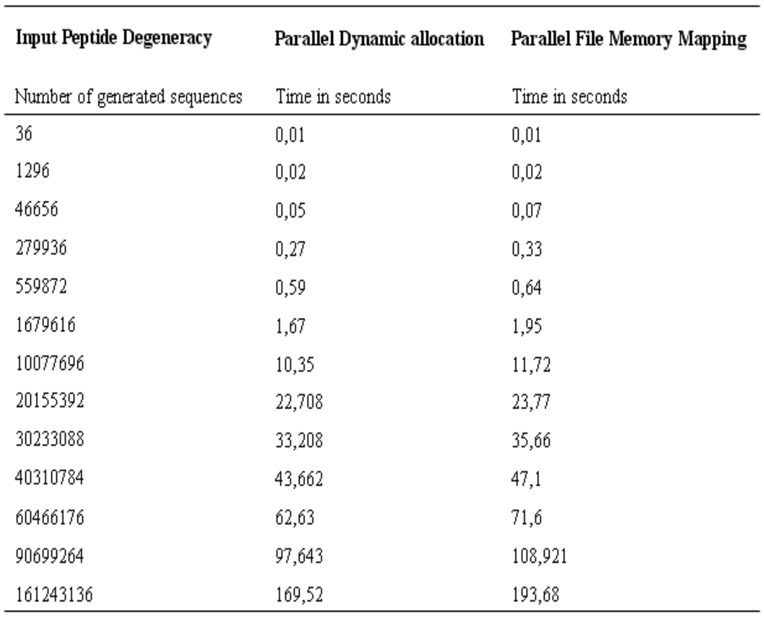
Even id a sequential approach is satisfactory, a parallelization process is interesting to to optimise the performances of our algorithms. A local cluster architecture is perfectly adapted to this kind of application. In the following section, we describe adaptations of the algorithms which optimize the resources of a cluster computing environment. This approach achieve high throughput non redundant protein to DNA backtranslation, it reduces the whole execution time and the results are dispatched on several disks to avoid disk space overloading.

C. *The computing cluster architecture*

A computing cluster is an homogeneous systolic architecture, which uses the power of interconnected machines to perform time consuming tasks. We have used the following cluster architecture: a system of 15 PCs (one master with an Intel Xeon, 2.40 GHz bi-processor; 2GB RAM and 14 worker nodes with the same configuration PCs are running under Linux and the networking is assured by a 1 GB/s Ethernet switch. Communications are mainly generated by the network file system (NFS). Although there are file systems specifically designed for PC clusters (Wang et al., 2002) the NFS was considered to be efficient enough for the purpose of this work. The main distributions of Linux include tools for controlling a parallel execution on PC clusters [4] [5]. For the application development we used the MPI library MPICH, v.1.1.2 with version 2.96 of the MPICC compiler (http://www-unix.mcs.anl.gov/mpi/). *Parallelization strategy*

To have an efficient parallelization for the two strategies – dynamic memory allocation and file memory mapping – we opted for a load balancing approach and a distribution of output files on all used nodes. In fact, each process (assigned to a node) computes a part of the whole result for a given peptide and locally save fragment results. Fragments are equally distributed on the cluster in order to have maximal performance and storage balancing on each node. MPI was used to dispatch data on all nodes and to subdivide computing for each process. Table 2 shows the execution time of the two parallel strategies

TABLE 2. PERFORMANCE DATA COMPARISON BETWEEN THE TWO PARALLEL ALGORITHMS.



Input Peptide Degeneracy	**Parallel Dynamic allocation**	**Parallel File Memory Mapping**
Number of generated sequences	Time in seconds	Time in seconds
36	0,01	0,01
1296	0,02	0,02
46656	0,05	0,07
279936	0,27	0,33
559872	0,59	0,64
1679616	1,67	1,95
10077696	10,35	11,72
20155392	22,708	23,77
30233088	33,208	35,66
40310784	43,662	47,1
60466176	62,63	71,6
90699264	97,643	108,921
161243136	169,52	193,68

Memory allocation specifically uses system memory but the memory mapping strategy only uses CPUs and IOs. Our algorithms construct for each node their own fragment structures by the two approaches and then perform locally the appropriate computation for each fragment oligos. To have such results, we split output data into 14 structures because we have 14 nodes; each structure contains the ordered oligos. For example, the oligo numbered 'n' is stored on node number 'x' and the oligo numbered 'n+1' is stored on node number 'x+1'; if x = 14 then oligo number n+1 is stored on node number 1 and so on until the last oligo. Moreover, each amino acid is processed as in the sequential algorithms. As mentioned in the previous section, the execution time for the sequential algorithms increases exponentially when input peptide degeneracy exceeds 3.10^6 oligos; but, with parallel approaches, we obtain the histograms shown in figure 6.

Parallel implementation shows a decrease in the whole execution time compared to sequential algorithms for the two approaches. We can even see a linear increase for the dynamic memory allocation up to degeneracy of 6^{10}. Dynamic memory allocation is more advantageous for small degeneracy peptides ($< 6^{11}$). Memory mapping execution time increases faster than dynamic memory allocation because the CPUs are used every time for computing codons and file pointer position. The algorithm using memory allocation has been parallelized by sharing memory and output data between Cluster nodes.

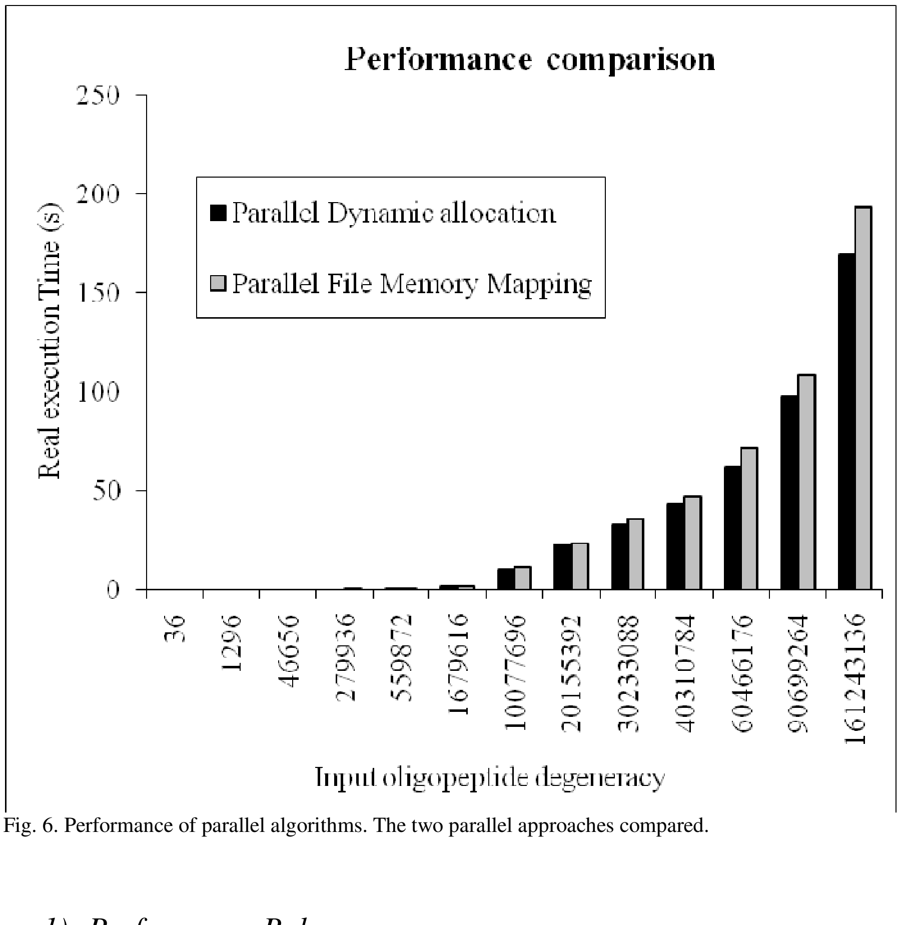


Fig. 6. Performance of parallel algorithms. The two parallel approaches compared.

1) Performance Balance

Using a cluster framework to compute oligos generated from a peptide backtranslation by parallel programming with MPI shows an improvement of the whole performance. In fact, such a distributed system offers multiple resources to compute a large number of oligos efficiently. Figure 7 shows that the time gain is enhanced when degeneracy increases if we use a parallel backtranslation on a computer cluster. We can observe that the file memory mapping approach is more efficient in the case of the sequential algorithms but, in the parallel version, we can see a better behaviour of parallel dynamic allocation. Performance declines when we try to find long oligos with high degeneracy (50 bases and more). We surpass system capacities: CPU, memory, and disk space when we work with oligopeptides of 30 amino acids and a degeneracy greater than 6^{13}.

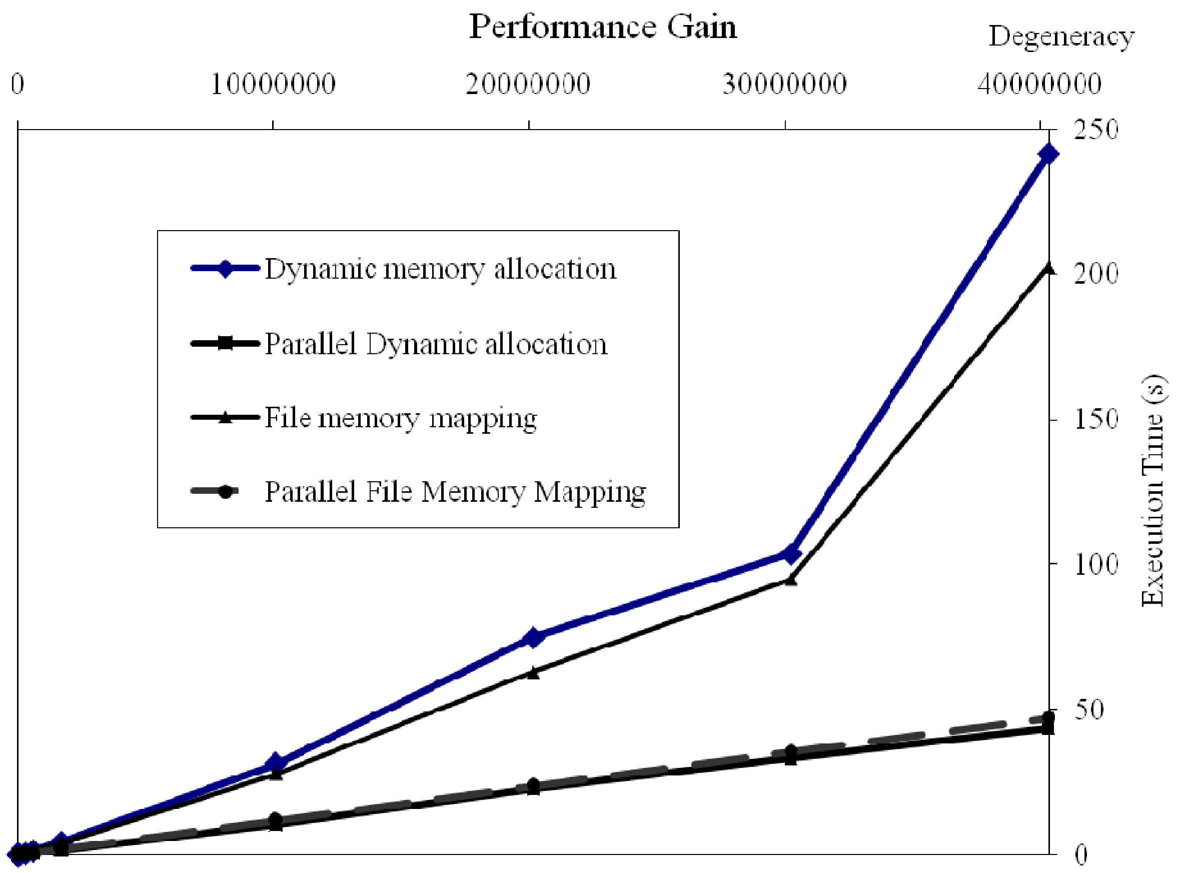


Fig. 7. Parallel vs. sequential backtranslation programs. The parallel versions of our algorithm shows a liner like curve for degeneracy smaller than 4.107.

In addition, Figure 8 evidences the relevance of parallel codes for such work. It shows that when we use a cluster architecture algorithm to backtranslate a 6^9 degeneracy oligopeptide, performance increases with the two previously mentioned strategies. In fact, the whole execution time decreases when algorithms use more and more nodes. The complexity remains exponential, but it can be significantly decreased by the number of computing nodes.

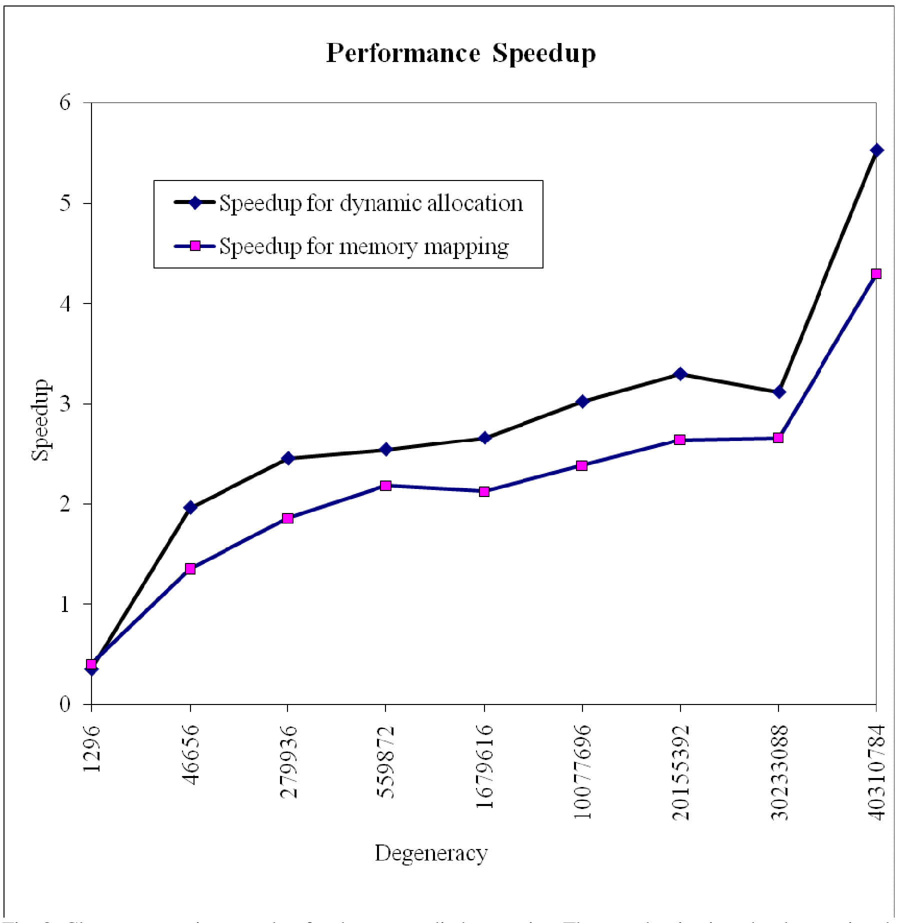


Fig. 8. Cluster computing speedup for the two studied strategies. The speedup is given by the quotient between the sequential execution time and the parallel one.

When we work on short oligopeptides we cannot distinguish the difference between parallel and sequential algorithms. Indeed, the oligpeptide length has an incidence on the whole execution time, as for sequential programs(when the length increases, the performance logically decreases).

IV. Conclusion

This paper introduces a novel approach giving a full backtranslation of oligopeptides and brings to light the utility of such a method in enzyme specific oligos design. This approach is able to produce all possible non degenerate DNA sequences corresponding to an input oligopeptide of reasonable degeneracy. This problem presents an exponential complexity and we have presented a new algorithm implemented with the C programming language with two different strategies minimising the execution time. In addition, we propose a parallel version of the two strategies developed using MPI in a computing cluster environment. The parallelization helps in dividing the algorithm complexity which remains however exponential. Performance tests indicate that the parallelization has clear advantages because it allows the fragmentation of the output oligos file knowing that a full backtranslation need significant disk space.

ACKNOWLEDGEMENTS

This work was financially supported by the regional council of Auvergne (PREVOIR and INSTRUIRE projects) and by a grant from the ACI Non Pollution-Dépollution, « Oxygen » project. M. Missaoui was supported by a grant from the regional council of Auvergne and C. Militon by a grant from “Ministère de l’éducation, de la recherche et de la technologie”. The authors thank Pr. Philip Hoggan for his helpful advices.

REFERENCES

[1] S. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of molecular biology,* vol. 215, pp. 403-10, October 5 1990

[2] O. R. Bininda-Emonds, "transAlign: using amino acids to facilitate the multiple alignment of protein-coding DNA sequences," *BMC Bioinformatics,* vol. 6, p. 156, june 22 2005.

[3] L. Bodrossy and A. Sessitsch, "Oligonucleotide microarrays in microbial diagnostics," *Current Opinion in Microbiology* vol. 7, pp. 245-254, May 14 2004.

[4] C. Bookman, *Linux Clustering: Building and Maintaining Linux Clusters*. USA: New Riders Publishing, 2002.

[5] I. Campbell, *Linux: Assuring High Availability*. USA: John Wiley & Sons, 2001.

[6] R. Chenna, H. Sugawara, T. Koike, R. Lopez, T. J. Gibson, D. G. Higgins, and J. D. Thompson, "Multiple sequence alignment with the Clustal series of programs," *Nucleic Acids Research,* vol. 31, pp. 3497-3500, 2003.

[7] R. M. Lewis, "PROBFIND: a computer program for selecting oligonucleotide probes from peptide sequences," *Nucleic Acids Research,* vol. 14, pp. 567-70, January 10 1986.

[8] J. Ma, T. Zhou, W. Gu, X. Sun, and Z. Lu, "Cluster analysis of the codon use frequency of MHC genes from different species," *Biosystems,* vol. 65, pp. 199-207 March 5 2002.

[9] N. Melab, S. Cahon, and E. Talbi, "Grid computing for parallel bio inspired algorithms," *Journal of parallel and Distributed Computing,* vol. 66 pp. 1052-1061, August 2006.

[10] A. Moreira and A. Maass, "TIP: protein backtranslation aided by genetic algorithms," *Bioinformatics,* vol. 20, pp. 2148-2149 April 1 2004.

[11] S. Moretti, F. Reinier, O. Poirot, F. Armougom, S. Audic, V. Keduas, and C. Notredame, "PROTOGENE: turning amino acid alignments into bona fide CDS nucleotide alignments," *Nucleic Acids Research,* vol. 34, pp. W600-W603, March 20 2006

[12] J. Nash, "A computer program to calculate and design oligonucleotide primers from amino acid sequences," *Computer applications in the biosciences,* vol. 9, pp. 469-71, August 1993.

[13] M. W. Nirenberg, "Historical review: Deciphering the genetic code--a personal account," *Trends in biochemical sciences* vol. 29, pp. 46-54, january 2004.

[14] G. Pesole, M. Attimonelli, and S. Liuni, "A backtranslation method based on codon usage strategy," *Nucleic Acids Research,* vol. 16, pp. 1715-1728, November 15 1988.

[15] S. Rimour, D. Hill, C. Militon, and P. Peyret, "GoArrays: highly dynamic and efficient microarray probe design," *Bioinformatics,* vol. 21, pp. 1094-1103 2005.

[16] P. Stothard, "The sequence manipulation suite: JavaScript programs for analyzing and formatting protein and DNA sequences," *Biotechniques,* vol. 28, pp. 1102, 1104, June 2000.

[17] M. Suyama, D. Torrents, and P. Bork, "PAL2NAL: robust conversion of protein sequence alignments into the corresponding codon alignments," *Nucleic Acids Research,* vol. 34, pp. W609-W612, April 11 2006.

[18] T. Tamura, S. Holbrook, and S. Kim, "A Macintosh computer program for designing DNA sequences that code for specific peptides and proteins," *Biotechniques,* vol. 10, pp. 782-4, 1991.

[19] J. Wang and Z. Xu, "Cluster file systems: a case study," *Future Generation Computer Systems,* vol. 18, pp. 373-387, 2002.

[20] R. Wernersson and A. G. Pedersen, "RevTrans: multiple alignment of coding DNA from aligned amino acid sequences," *Nucleic Acids Research,* vol. 31, pp. 3537-3539, April 14 2003.
[21] G. White and W. Seffens, "Using a neural network to backtranslate amino acid sequences," *Electronic Journal of Biotechnology,* vol. 1, pp. 17-18, December 1998.
[22] WebLab: backtranseq: http://weblab.cbi.pku.edu.cn/program.inputForm.do?program=backtranseq
[23] CLC Protein Workbench: http://www.clcbio.com/index.php?id=93
[24] Gene Design - Reverse translation: http://slam.bs.jhmi.edu/cgi-bin/gd/gdRevTrans.cgi