



HOKKAIDO UNIVERSITY

Title	Large-scale search for genes on which positive selection may operate
Author(s)	Endo, Toshinori
Degree Grantor	The Graduate University for Advanced Studies
Degree Name	博士(理学)
Issue Date	1995
Doc URL	https://hdl.handle.net/2115/49989
Type	doctoral thesis
File Information	thesis_endot.pdf



Large-Scale Search for Genes on Which Positive Selection May Operate

Toshinori Endo

Doctor of Philosophy

Department of Genetics

School of Life Science

Graduate University for Advanced Studies

1995

Acknowledgments

I would like to express my gratitude to Professor Takashi Gojobori for his guidance and support and for sharing his ideas about molecular evolution. I also would like to thank Professor Yoshio Tateno and Dr. Naruya Saitou for teaching me about molecular phylogeny and molecular evolution and their helpful suggestions. I would appreciate Professor Tomoko Ohta, Hiroshi Hori and Toshimichi Ikemura for their comments, suggestions and discussions. I would express my gratitude to my parents for their support and encouragement throughout my studies. I would thank all the DDBJ staffs for their warm heart and patience by allowing me use of the super computer, CRAY S-2000, for months, which sometimes interfered DDBJ activities by too much load caused by this study.

Abstract

There has been great controversy between neutralists and selectionists concerning what proportion of mutants contributed to gene evolution as advantageous mutants. Several studies have shown that there are some genes on which positive selection operates through comparison and statistical analysis of nucleotide sequences, but no one have been able to estimate the proportion of genes on which positive selection operates among all the genes available. One reason is that conducting such research involves a large amount of DNA sequence data and coordinated analysis in order to conduct large-scale searches for candidate genes making full use of sophisticated computers. The purpose of this study was to investigate the role of positive selection in the course of gene evolution. For the first attempt, I estimated the proportion of genes on which positive selection may operate utilizing a large-scale search.

In PART I, I discuss the strategy and results of a large-scale search for genes on which positive selection may operate. In this study, a large-scale search is defined as search for genes which can fulfill certain criteria by utilizing a large amount of DNA sequence data using the computers of high level processibility including a super computer. The search was performed

using the following strategies: 1) alignment construction for the 3,595 groups of homologous sequences of non-partial coding sequences available in the DDBJ DNA Database Release 11; 2) comparison of numbers of synonymous and nonsynonymous substitutions. From the search, 17 homologous gene groups were obtained as candidate genes on which positive selection may operate. They constituted 0.47% of all the gene groups examined in the study. Interestingly enough, nine out of the 17 gene groups were for surface proteins of viruses and parasites. A window analysis of numbers of synonymous and nonsynonymous substitutions was performed to locate possible within-gene regions subject to positive selection. The analysis indicated that a region where the number of nonsynonymous substitutions was larger than that of synonymous substitutions corresponded to a known antigenic epitope region.

In PART II, an additional search was conducted to find genes which contained a region or a segment subject to positive selection. This search was performed because the nucleotide sites responsible for positive selection usually are not spread throughout the entire gene even when positive selection is operating. This was the cause of my inability to locate MHC genes in the search in PART I, because positive selection is shown to be operating only on the antigenic sites in MHC genes. In the new search shown in part II, 192 gene groups were obtained as candidate genes having within-gene regions

where positive selection may operate. They constituted about five percent of all the genes examined.

Table of Contents

ACKNOWLEDGMENTS.....	II
ABSTRACT	III
TABLE OF CONTENTS	VI
LIST OF TABLES.....	VIII
LIST OF FIGURES.....	IX
LIST OF ABBREVIATIONS.....	X
 PART I LARGE-SCALE SEARCH FOR GENES ON WHICH POSITIVE SELECTION MAY OPERATE	 13
Introduction	13
Materials and Methods	16
<i>Construction of coding sequence database</i>	<i>16</i>
<i>The database for homologous amino acid sequences</i>	<i>17</i>
<i>Back translation of alignments of amino acid sequence into alignments of DNA sequences</i>	<i>18</i>
<i>Search for candidate genes by comparing d_N with d_S for whole coding sequences</i>	<i>18</i>
<i>Window analysis for d_S and d_N of candidate genes</i>	<i>19</i>
Results and Discussion	21
<i>Ratio of candidate genes on which positive selection may operate</i>	<i>21</i>
<i>Members of candidate genes.....</i>	<i>21</i>
<i>Identification of the within-gene region where positive selection operates</i>	<i>23</i>
(i) The case of merozoite surface antigen 2 genes of malaria.....	23
(ii) Case of the major surface protein 1 α of rickettsia	26
<i>Other genes.....</i>	<i>27</i>
<i>Positive selection seems to operate on only a small proportion of genes.....</i>	<i>39</i>
<i>Common features among candidate genes</i>	<i>41</i>
Summary	43
 PART II POSITIVE SELECTION MAY OPERATE ON 5% OF THE GENES SEQUENCED SO FAR	 45
Introduction	45
Materials and methods	47
<i>Window search method.....</i>	<i>47</i>
<i>Statistical Test.....</i>	<i>48</i>
Results and Discussion.....	49
<i>About 5% of genes may be subject to positive selection</i>	<i>49</i>

<i>The characteristics of gene groups</i>	49
<i>Gene structure and positive selection</i>	51
<i>Kinds of candidate genes on which positive selection may operate</i>	51
<i>Bias of DNA sequence database</i>	52
<i>G+C content of candidate genes</i>	54
<i>Genes for surface proteins of parasites and viruses</i>	54
<i>Genes for secretory proteins</i>	55
<i>Genes for toxin genes</i>	55
<i>Cell adhesion molecules and cell surface antigens of non-parasitic genes</i>	56
<i>Enzyme genes</i>	56
<i>Common features among candidate genes containing within-gene regions where positive selection may operate</i>	57
Summary	58
 REFERENCES	 72
PART I.....	72
PART II.....	76
 APPENDIX	 77
COMPUTER PROGRAMS	77
bstn, bstn (Perl): perform bootstrap resampling of amino acid (bstn) or nucleotide (bstn) sequence and summarize the result.	77
cds1.y, cds2.y (yacc): to extract coding sequences from DDBJ/GenBank formatted database WITH (cds1) or WITHOUT (cds2) length checking.	82
aln (perl) - reformat alignment.	91
dall.c (C): create coding sequence alignments using SODHO database and coding sequence database.	92
distng.c (C): calculating the numbers of synonymous and nonsynonymous substitutions between pairs of sequence properly aligned.	101
gettrans (Perl): get translation feature from GenBank formatted database.	109
GREP (Perl): find the double slash-separated records that contains on or more keywords.	111
mkidx.c (C) - makes index file for cDNA database.	112
pos (Perl): find the within-gene regions where positive selection may operate.....	113
posblk (Perl): find the within-gene regions on which positive selection may operate..	118
possig (Perl) - searches for the gene regions where positive selection may operate with examining statistical significance.	121
reduce (Perl): remove excess gaps and redundant sequences within alignments.	125
reformat (Perl) - reformat alignment.....	127
separate (Perl) - separate alignment into individual sequences.	129
seq (Perl): reformat sequence file.....	130
SORT (Perl) - sort records.	136
wal (Perl) - widow analysis.....	137
wp1 (Perl) - plots the result of wal.....	139

List of Tables

TABLE 1. GENE GROUPS ON WHICH POSITIVE SELECTION MAY OPERATE.	22
TABLE 2. CATEGORIES OF THE PRODUCTS OF THE CANDIDATE GENE.	50
TABLE 3. CANDIDATE GENES THAT CONTAIN REGIONS POSSIBLY SUBJECT TO POSITIVE SELECTION	59

List of Figures

FIGURE 1. SEARCH STRATEGIES FOR GENES ON WHICH POSITIVE SELECTION OPERATES.	20
FIGURE 2. WINDOW ANALYSIS OF MEROZOITE SURFACE ANTIGEN 2 (<i>MSA2</i>) GENE OF <i>PLASMODIUM FALCIPARUM</i>	25
FIGURE 3. THE GENE STRUCTURE OF <i>MSA2</i>	25
FIGURE 4. WINDOW ANALYSIS OF MAJOR SURFACE PROTEIN 1A (<i>MSP1A</i>) GENE OF <i>ANAPLASMA</i> <i>MARGINALE</i>	26
FIGURE 5. GENE STRUCTURE OF <i>MSP1A</i>	27
FIGURE 6. <i>CHLAMYDIA</i> OUTER MEMBRANE PROTEIN.	31
FIGURE 7. EQUINE INFECTIOUS ANEMIA VIRUS ENVELOPE PROTEIN.	32
FIGURE 8. <i>PSEUDORABIES VIRUS GH</i> GENE.	32
FIGURE 9. BACTERIOPHAGE <i>E</i> GENE.	33
FIGURE 10. REOVIRUS SIGMA-1 PROTEIN.	33
FIGURE 11. <i>SHIGELLA</i> INVASION PLASMID ANTIGEN IPAC.	34
FIGURE 12. <i>SHIGELLA</i> INVASION PLASMID ANTIGEN IPAD.	34
FIGURE 13. <i>APLYSIA CALIFORNICA</i> EGG-LAYING HORMONE.	35
FIGURE 14. <i>APLYSIA CALIFORNICA</i> EGG-LAYING HORMONE A.	35
FIGURE 15. <i>ESCHERICHIA COLI</i> ATP SYNTHASE F ₀ SUBUNIT.	36
FIGURE 16. <i>ESCHERICHIA COLI</i> NEOMYCIN RESISTANCE PROTEIN.	36
FIGURE 17. <i>YERSINIA</i> ADHESION MOLECULE.	37
FIGURE 18. RAT PROSTATIC STEROID BINDING PROTEIN.	37
FIGURE 19. SNAKE NEUROTOXINS.	38
FIGURE 20. <i>SACCHAROMYCES CEREVISIAE</i> CDC6.	38
FIGURE 21. WINDOW ANALYSIS OF A LOCUS OF HUMAN CLASS I MHC GENES.	40
FIGURE 22. THE WINDOW SEARCH METHOD.	53
FIGURE 24. G+C CONTENT OF ALL GENES EXAMINED IN THE ANALYSIS.	53

List of Abbreviations

General

d_N	The number of nonsynonymous substitutions.
d_S	The number of synonymous substitutions.
CD5	<u>cluster of differentiation 5</u>
<i>cdc6</i>	<u>cell division cycle 6</u>
EB virus	<u>Epstein-Barr virus</u>
EIAV	<u>equine infectious anemia virus</u>
<i>env</i>	<u>envelope</u>
<i>gag</i>	gene for group-specific <u>antigen</u> , or capsid precursor gene
GCN2	yeast general <u>amino acid control 2</u>
gp195	glycoprotein
HBV	<u>hepatitis B virus</u>
HCV	<u>hepatitis C virus</u>
HIV1	<u>human immunodeficiency virus type 1</u>
HSV	<u>herpes simplex virus</u>
HTLV-I	<u>human T-cell leukemia virus type I</u>
<i>ipaC</i>	<u>invasion plasmid antigen C</u>
ORF	<u>open reading frame</u>
LMP1	<u>latent membrane protein 1</u>
LTR	<u>long terminal repeat</u>
MHC	<u>major histocompatibility complex</u>
MSA2	<u>Merozoite Surface Antigen 2</u>
<i>msp1α</i>	<u>major surface protein 1α</u>
<i>omp</i>	<u>outer membrane protein</u>
<i>pol</i>	<u>RNA dependent DNA polymerase</u>
RSV	<u>Rous sarcoma virus</u>
$V(X)$	Variance of probability variable X.
VP1	<u>viral protein 1</u>
<i>yadA</i>	<u>Yersinia adhesion molecule A</u>

Species names

<i>A. calcoaceticus</i>	<i>Acinetobacter calcoaceticus</i>
<i>A. californica</i>	Sea snail <i>Aplysia californica</i>
<i>A. chroococcum</i>	<i>Azotobacter chroococcum</i>
<i>A. eutrophus</i>	<i>Alcaligenes eutrophus</i>
<i>A. hydrophila</i>	<i>Aeromonas hydrophila</i> ビアリオ科 17-5 陰性 嫌気性 細菌
<i>A. lupus</i>	<i>Anarhichas lupus</i>
<i>A. marginale</i>	Rickettsia <i>Anaplasma marginale</i>
<i>A. nidulans</i>	<i>Azotobacter nidulans</i>
<i>A. rhizogenes</i>	<i>Agrobacterium rhizogenes</i>
<i>A. niger</i>	<i>Aspergillus niger</i>
<i>A. tumefaciens</i>	<i>Agrobacterium tumefaciens</i>
<i>A. vinelandii</i>	<i>Azotobacter vinelandii</i>

<i>B. bronchiseptica</i>	<i>Bordetella bronchiseptica</i>
<i>B. parapertussis</i>	<i>Bordetella parapertussis</i>
<i>B. paraspontia</i>	
<i>B. pertussis</i>	<i>Bordetella pertussis</i>
<i>B. taurus</i>	<i>Bos taurus</i>
<i>C. biennis</i>	<i>Choristoneura biennis</i>
<i>C. botulinum</i>	<i>Clostridium botulinum</i>
<i>C. difficile</i>	<i>Clostridium difficile</i>
<i>C. fimi</i>	<i>Cellulomonas fimi</i>
<i>C. gladiata</i>	<i>Canavalia gladiata</i>
<i>C. pasteurianum</i>	<i>Clostridium pasteurianum</i>
<i>C. reinhardtii</i>	<i>Calabaria reinhardtii</i>
<i>C. reinhardtii</i>	<i>Chlamydomonas reinhardtii</i>
<i>C. tetani</i>	<i>Clostridium tetani</i>
<i>C. vicina</i>	<i>Calliphora vicina</i>
<i>D. angusticeps</i>	Sea snake <i>Dendroaspis angusticeps</i>
<i>D. grimshawi</i>	<i>Drosophila grimshawi</i>
<i>D. melanogaster</i>	<i>Drosophila melanogaster</i>
<i>D. subobscura</i>	<i>Drosophila subobscura</i>
<i>D. virilis</i>	<i>Drosophila virilis</i>
<i>E. coli</i>	colon bacterium <i>Escherichia coli</i>
<i>H. corrugata</i>	<i>Haliotis corrugata</i>
<i>H. influenzae</i>	<i>Haemophilus influenzae</i>
<i>H. gallinarum</i>	<i>Haemophilus gallinarum</i>
<i>H. halobium</i>	<i>Halobacterium halobium</i>
<i>H. rufescens</i>	<i>Haliotis rufescens</i>
<i>H. volcanii</i>	<i>Halobacterium volcanii</i>
<i>K. pneumoniae</i>	<i>Klebsiella pneumoniae</i>
<i>L. donovani</i>	<i>Leishmania donovani</i>
<i>L. chagasi</i>	<i>Leishmania chagasi</i>
<i>L. semifasciata</i>	sea snake <i>Laticauda semifasciata</i>
<i>M. americanus</i>	<i>Macrozoarces americanus</i>
<i>M. capsulatus</i>	<i>Methylobacter capsulatus</i>
<i>M. capsulatus</i>	<i>Methylococcus capsulatus</i>
<i>M. fascicularis</i>	<i>Macaca fascicularis</i>
<i>M. musculus</i>	<i>Mus musculus</i>
<i>M. xanthus</i>	<i>Myxococcus xanthus</i>
<i>N. alata</i>	<i>Nicotiana alata</i>
<i>N. crassa</i>	<i>Neurospora crassa</i>
<i>N. gonorrhoeae</i>	<i>Neisseria gonorrhoeae</i>
<i>N. meningitidis</i>	<i>Neisseria meningitidis</i>
<i>N. tabacum</i>	<i>Nicotiana tabacum</i>
<i>P. aeruginosa</i>	<i>Pseudomonas aeruginosa</i>
<i>P. asaccharolyticus</i>	<i>Peptostreptococcus asaccharolyticus</i>
<i>P. chrysogenum</i>	<i>Penicillium chrysogenum</i>
<i>P. denitrificans</i>	<i>Paracoccus denitrificans</i>
<i>P. denitrificans</i>	<i>Pseudomonas denitrificans</i>
<i>P. diminuta</i>	<i>Pseudomonas diminuta</i>
<i>P. denitrificans</i>	<i>Pseudomonas denitrificans</i>
<i>P. falciparum</i>	Malaria parasite <i>Plasmodium falciparum</i>
<i>P. glauca</i>	<i>Phaseolus glauca</i>

<i>P. inflata</i>	<i>Petunia inflata</i>
<i>P. oleovorans</i>	<i>Pseudomonas oleovorans</i>
<i>P. putida</i>	<i>Pseudomonas putida</i>
<i>P. vulgaris</i>	<i>Phaseolus vulgaris</i>
<i>R. capsulata</i>	<i>Rhodobacter capsulata</i>
<i>R. capsulata</i>	<i>Rhodopseudomonas capsulata</i>
<i>R. prowazekii</i>	<i>Rickettsia prowazekii</i>
<i>R. rickettsii</i>	<i>Rickettsia rickettsii</i>
<i>R. rubrum</i>	<i>Rhodospirillum rubrum</i>
<i>R. sphaeroides</i>	<i>Rhodobacter sphaeroides</i>
<i>R. sphaeroides</i>	<i>Rhodopseudomonas sphaeroides</i>
<i>R. viridis</i>	<i>Rhodopseudomonas viridis</i>
<i>S. aurantiaca</i>	<i>Stigmatella aurantiaca</i>
<i>S. aureus</i>	<i>Staphylococcus aureus</i>
<i>S. cerevisiae</i>	Budding Yeast <i>Saccharomyces cerevisiae</i>
<i>S. coelicolar</i>	<i>Streptomyces coelicolar</i>
<i>S. curacoi</i>	<i>Streptomyces curacoi</i>
<i>S. erythraea</i>	<i>Saccharopolyspora erythraea</i>
<i>S. dysenteriae</i>	<i>Shigella dysenteriae</i>
<i>S. flexneri</i>	<i>Shigella flexneri</i>
<i>S. fradiae</i>	<i>Streptomyces fradiae</i>
<i>S. glaucescens</i>	<i>Streptomyces glaucescens</i>
<i>S. griseus</i>	<i>Streptomyces griseus</i>
<i>S. lividans</i>	<i>Streptomyces lividans</i>
<i>S. oleracea</i>	<i>Spinacia oleracea</i>
<i>S. pombe</i>	Fission Yeast <i>Shizosaccharomyces pombe</i>
<i>S. pyogenes</i>	<i>Streptococcus pyogenes</i>
<i>S. ribosidifucus</i>	<i>Streptomyces ribosidifucus</i>
<i>S. scrofa</i>	pig <i>Sus scrofa</i>
<i>S. thermotolerans</i>	<i>Streptomyces thermotolerans</i>
<i>S. violaceoruber</i>	<i>Streptomyces violaceoruber</i>
<i>T. brucei</i>	<i>Trypanosoma brucei</i>
<i>T. cruzi</i>	<i>Trypanosoma cruzi</i>
<i>T. equiperdum</i>	<i>Trypanosoma equiperdum</i>
<i>U. maydis</i>	<i>Ustilago maydis</i>
<i>V. cholerae</i>	<i>Vibrio cholerae</i>
<i>V. faba</i>	<i>Vicia faba</i>
<i>X. campestris</i>	<i>Xanthomonas campestris</i>
<i>X. flavus</i>	<i>Xanthobacter flavus</i>
<i>X. laevis</i>	<i>Xenopus laevis</i>
<i>Y. enterocolitica</i>	<i>Yersinia enterocolitica</i>
<i>Y. pseudotuberculosis</i>	<i>Yersinia pseudotuberculosis</i>

Xanthomonas & *Pseudomonadaceae*

Synonyms of animal names

avian	bird	murine	mouse
bovine	cow	ovine	sheep
canine	dog	porcine	pig
equine	horse	swine	pig
feline	cat		

PART I

Large-Scale Search for Genes on Which Positive Selection May Operate

Introduction

The neutral theory of molecular evolution maintains that fixation of neutral mutations by random genetic drift within a population is the basis of evolution of most genes (Kimura 1983). Adaptive mutations can contribute to gene evolution more directly than neutral mutations, though they may occur at an extremely low frequency. Most deleterious mutations, on the other hand, do not contribute to gene evolution because they have a great chance of being lost within a small number of generations even though they may occur at a high frequency.

The genes on which positive selection operates are considered to have an evolutionary characteristic in which the number (d_N) of nonsynonymous

substitution is larger than that (d_S) of synonymous substitutions, because positive selection favors the nonsynonymous substitution which causes adaptive amino acid changes rather than the synonymous substitution which does not cause any amino acid changes. By adopting this characteristic as the criterion, it was been shown that the highly polymorphic genes for major histocompatibility complex (MHC) classes I and II, may be subjected to positive selection due to the fact that d_S is significantly larger than d_N at antigen recognition sites in the genes (Hughes and Nei 1988, 1989). Similarly, several other genes can be considered candidate genes on which positive selection may operate, such as the stomach lysozyme genes of ruminants (Irwin and Wilson 1990), visual pigment genes (Yokoyama and Yokoyama 1990), homeobox genes (Kappen et al. 1989), gamma hemoglobin (Fitch et al. 1991), alcohol dehydrogenase genes (Long and Langley 1993), ion channel genes (Strong et al. 1993), growth hormone genes (Ohta 1993), α_1 -antitrypsin genes (Ohta 1994), and abalone sperm lysin genes (Youn-Ho Lee et al. 1995). Since these studies were conducted on the individual cases, the proportion of such genes on which positive selection operates has been previously unknown. Thus, it is very important to understand what proportion of genes are affected by positive selection and the identity of these genes, in order to understand not only the evolutionary mechanisms of the genes but also the

relationship between the evolutionary process and the functions of those genes.

With the aim of answering these intriguing questions, I developed a new method to facilitate a systematic search for candidate genes from the DNA sequence database on which positive selection may operate. In this method, I created a multiple alignment database of 3,595 homologous cDNA sequences, and estimated the numbers of synonymous and nonsynonymous substitutions among all possible pairs in all homologous DNA sequences.

Materials and Methods

Construction of coding sequence database

The coding sequence database was constructed from a total of 57,741 entries of DNA sequences in DNA Database DDBJ Release 11 which included the databases DDBJ, EMBL and GenBank. In the feature table of entries, coding region is represented as follows:

```
CDS      262..1434
          /note="transforming growth factor-beta precursor"
          /codon_start=1
CDS      join(671..857,1607..1752)
          /product="Preproinsulin"
          /codon_start=1
CDS      join(complement(7420..7474),complement(7161..7284),
          complement(5203..5326))
          /note="preapolipoprotein C-III"
          /codon_start=1
CDS      complement(join(10731..10787,complement(9299..9965),
          7559..9162))
          /note="immediate early transcriptional activator IE110"
CDS      <3..1232
          /note="alpha-1-antitrypsin precursor"
          /codon_start=1
CDS      1..849
          /product="calpastatin"
          /gene="calpastatin"
          /codon_start=1
          /partial
```

where the terms mean:

CDS	coding region
join	joining more than two fragments of sequence
complement	the reverse-complement sequence of the region
/partial	partial sequence
/pseudo	pseudogene

To isolate coding regions and join them into a single cDNA sequence from

these complicated grammatical expressions, I developed a computer program named '*cds1*' using programming languages *C* and '*yacc*'. '*yacc*' stands for '*yet another compiler compiler*,' which was designed for creating compilers of programming languages (Aho and Ullman 1977). Description errors for coding regions in the database were carefully avoided by checking the following points: 1) the number of nucleotides should be a multiple of three; 2) initiation codons must be ATG, CTG, GTG or TTG; and 3) no termination codons should be contained within the described frames. I also excluded partial sequences and pseudogenes from the present study by referring to the feature table in entries of the database. As a result, nucleotide sequences for 24,832 complete coding regions were obtained from all the entries in the database.

The database for homologous amino acid sequences

All the coding sequences extracted by *cds1* were translated into the amino acid sequences. The amino acid sequences obtained were then categorized into an appropriate number of gene groups of homologous sequences by using the computer program *BLAST*, which was designed for homology search (Altschul et al. 1990). I set 0.5 as the maximum threshold of expected probability score in the homology search. Homologies of the

sequence groups were then examined using the algorithm of global homology search originally developed by Needleman and Wunsch (Needleman and Wunsch 1970). I used only the sequences which had a similarity score larger than 25% of the score of the original sequence. These homologous sequences were then multiply-aligned to create the database called "SODHO Ver.2" (Gojobori et al. 1993). Recently, SODHO Ver.3 was constructed by applying the similar procedures to DDBJ Release 20.

Back translation of alignments of amino acid sequence into alignments of DNA sequences

The multiple alignments of homologous amino acid sequences were converted into nucleotide sequence alignments by referring to the corresponding cDNA sequences. The set of resulting alignments was named "cDNA multiple alignments."

Search for candidate genes by comparing d_N with d_S for whole coding sequences

Candidate genes on which positive selection may operate were

searched for by comparing the estimated number (d_S) of synonymous substitutions with that (d_N) of nonsynonymous substitutions. The values of d_S and d_N were estimated by Nei and Gojobori's (1986) method for all sequence pairs in the cDNA multiple alignments for each gene group. If d_N 's were larger than the d_S 's for more than half of the sequence pairs compared in a particular gene group, the gene group was regarded as a candidate. I excluded gene groups in which either d_S or d_N was larger than 1 to avoid the saturation effect of nucleotide substitutions in the estimation process. I also inspected the resulting alignments manually to exclude such things as frame shift. The entire procedures for finding candidate genes are summarized in Figure 1.

Window analysis for d_S and d_N of candidate genes

In order to identify particular regions where positive selection may be operating within candidate genes, I conducted a window analysis.

A window was defined as a sequence region of with a 20-codon length on an alignment of homologous genes. In a window analysis, d_S and d_N are estimated for each window along the nucleotide sites, codon by codon, to test whether d_N is larger than d_S . If d_N is larger than d_S in the within-gene region, the region was considered to be subject to positive selection.

PART I A large-scale search for genes on which positive selection may operate

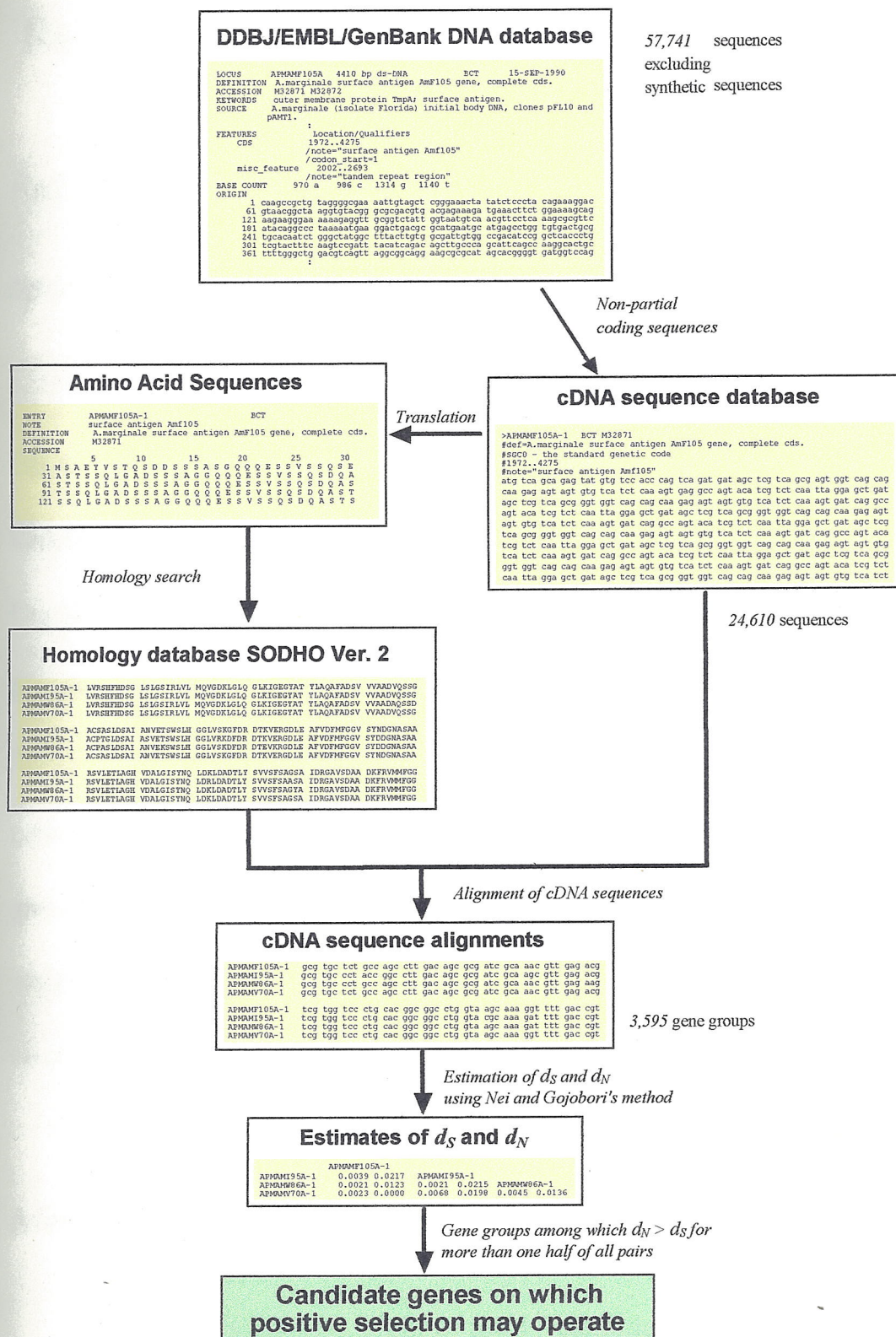


Figure 1. Search strategies for genes on which positive selection operates.

Results and Discussion

Ratio of candidate genes on which positive selection may operate

A total of 24,832 complete coding sequences were extracted from the DNA sequence database. All of those sequences were classified into 3,595 homologous gene groups except for 4,845 unclassifiable sequences. Unclassifiable sequences were defined as sequences that were not included in any of the gene groups of the cDNA multiple alignments. Surveying the 3,595 homologous gene groups, I searched for gene groups which met the criterion of d_N being larger than d_S for more than half of the pairs in each gene group. Finally, I obtained 17 gene groups considered as candidate genes on which positive selection may operate (Table 1). The candidate genes thus constituted 0.47% of all the gene groups examined.

Members of candidate genes

Interestingly enough, eight of the 17 candidate genes were antigenic surface proteins of parasites and viruses (Table 1). They included the outer membrane protein gene (*omp*) of *Chlamydia*, an envelope protein gene of the

equine infectious anemia virus, the merozoite surface antigen 2 (MSA2) gene of malaria parasite *Plasmodium falciparum*, the *E* gene of phages G4, FX174 and S13, the glycoprotein gene, *gH*, of pseudorabies virus, the *sigma-1* protein gene of the reovirus, the major surface protein (*msp1* α) of rickettsia *Anaplasma marginale*, and an invasion plasmid antigen of *Shigella*. These results might indicate that amino acid changes are taking place on these genes in order to allow the organisms to escape from the defensive system of the hosts, such as the immune system of vertebrates. I will discuss the characteristic features later for the remaining gene groups.

Table 1. Gene groups on which positive selection may operate.

Gene group	Representative species	n ¹⁾	d _S ²⁾	d _N ³⁾
merozoite surface antigen (MSA2) gene	<i>Plasmodium falciparum</i>	18	$1.1 \times 10^{-1} \pm 1.6 \times 10^{-2}$	$1.3 \times 10^{-1} \pm 1.2 \times 10^{-2}$
major surface protein (<i>msp1</i> α) gene	<i>Anaplasma marginale</i>	4	$5.4 \times 10^{-3} \pm 6.0 \times 10^{-6}$	$1.4 \times 10^{-2} \pm 5.8 \times 10^{-5}$
Outer membrane protein (<i>omp</i>) gene	<i>Chlamydia</i>	5	$3.6 \times 10^{-1} \pm 2.0 \times 10^{-1}$	$1.2 \times 10^{-1} \pm 1.8 \times 10^{-2}$
<i>env</i>	Equine infectious anemia virus	6	$1.0 \times 10^{-2} \pm 2.6 \times 10^{-5}$	$1.1 \times 10^{-2} \pm 1.8 \times 10^{-5}$
glycoprotein <i>gH</i> gene	Pseudorabies virus	3	$2.3 \times 10^{-3} \pm 6.5 \times 10^{-7}$	$3.0 \times 10^{-3} \pm 1.0 \times 10^{-7}$
<i>E</i> gene	phages G4, ϕ X174 and S13	4	$1.4 \times 10^{-1} \pm 8.4 \times 10^{-3}$	$1.7 \times 10^{-1} \pm 1.3 \times 10^{-2}$
<i>sigma-1</i> protein gene	Reovirus	4	$3.4 \times 10^{-3} \pm 1.2 \times 10^{-5}$	$7.7 \times 10^{-3} \pm 1.5 \times 10^{-5}$
invasion plasmid antigen gene (<i>ipaC</i>)	<i>Shigella</i>	3	0.0	$1.6 \times 10^{-3} \pm 1.2 \times 10^{-6}$
invasion plasmid antigen gene (<i>ipaD</i>)	<i>Shigella</i>	3	$2.0 \times 10^{-2} \pm 1.2 \times 10^{-4}$	$2.4 \times 10^{-2} \pm 2.6 \times 10^{-4}$
egg-laying hormone	<i>Aplysia californica</i>	6	$1.1 \times 10^{-1} \pm 5.1 \times 10^{-3}$	$1.6 \times 10^{-1} \pm 1.1 \times 10^{-2}$
egg-laying hormone A peptide	<i>Aplysia californica</i>	3	$8.3 \times 10^{-3} \pm 3.1 \times 10^{-5}$	$8.7 \times 10^{-3} \pm 1.8 \times 10^{-5}$
ATP synthase F ₀ subunit (<i>atp-2</i>) gene	<i>Escherichia coli</i>	4	$7.2 \times 10^{-3} \pm 2.6 \times 10^{-5}$	$9.7 \times 10^{-3} \pm 3.3 \times 10^{-5}$
Neomycin resistance protein gene	<i>Escherichia coli</i>	3	$1.8 \times 10^{-3} \pm 1.7 \times 10^{-6}$	$2.8 \times 10^{-3} \pm 1.0 \times 10^{-6}$
adhesion molecule gene (<i>yadA</i>)	<i>Yersinia</i>	3	$5.2 \times 10^{-2} \pm 2.5 \times 10^{-4}$	$6.7 \times 10^{-2} \pm 1.3 \times 10^{-4}$
prostatic steroid binding protein	Rat	7	$1.7 \times 10^{-1} \pm 1.5 \times 10^{-2}$	$1.9 \times 10^{-1} \pm 2.4 \times 10^{-2}$
neurotoxin	Snake	11	$2.8 \times 10^{-1} \pm 2.6 \times 10^{-2}$	$3.7 \times 10^{-1} \pm 2.2 \times 10^{-2}$
CDC6	<i>Saccharomyces cerevisiae</i>	3	$4.7 \times 10^{-3} \pm 1.1 \times 10^{-5}$	$8.9 \times 10^{-3} \pm 1.4 \times 10^{-5}$

¹⁾ The number of sequences. ²⁾ The number of synonymous substitutions per site. ³⁾ The number of nonsynonymous substitutions per site.

Identification of the within-gene region where positive selection operates

With the aim of locating regions where positive selection operates within candidate genes, I conducted a window analysis for all 17 candidate genes. I will use the results for two candidate genes as examples.

(i) The case of merozoite surface antigen 2 genes of malaria

The first example is the merozoite surface antigen 2 (*MSA2*) genes of malaria *P. falciparum*. Malaria is a well-known parasite that causes malarial fever. *MSA2* is known to be one of the genes expressed during asexual stages in malarial life cycle (Smythe et al. 1988). Merozoite is the stage of the malarial life cycle in which malaria is exposed to the human blood stream outside of the time of infection. Thus, it is quite possible that the genes expressed in the merozoite stage are targets of the immune attack from the human body. In fact, a high degree of polymorphism of *MSA2*, just like that in MHC genes, has been reported in various studies (Smythe et al. 1988, 1990, 1991; Elliott et al. 1990; Fenton et al. 1991; Marshall et al. 1991).

Figure 2 shows the results of the window analysis for two alleles of *MSA2*. The window size is twenty codon length. It is clear from the figure

that there are two regions (around 65-350 and 400-600) where the number of nonsynonymous substitutions is high and no synonymous substitution is observed. Although the function of the second region is not well understood, the first region corresponds exactly to an antigenic epitope already known (Figure 3; Fenton et al. 1991). This suggests that positive selection is operating on the antigenic epitopes of *MSA2* to facilitate escape from the host immune system. It also implies that the second region found in this analysis is another possible antigenic epitope.

It is very interesting that no synonymous substitutions were found over the entire region of *MSA2*. In the case of *MSA1*, another malaria surface protein gene, a small number of synonymous substitutions have been observed (Hughes 1991, 1992). One possible explanation is the short divergence time between the alleles. This explanation, however, is unlikely, because so many nonsynonymous substitutions were observed between the alleles. Another possibility is the reduction of synonymous substitutions. It is noteworthy that the genome of *P. falciparum* is extremely A+T rich, the G+C content being only 18% at the codon third positions (Musto et al. 1995). This implies that nucleotide substitutions occur mostly between A and T through some kind of mutation pressure. It is clear from the genetic codes that nucleotide changes between A and T alter coded amino acids even at the third

codon position except the four codon box. Whether nucleotide substitutions occur from A or T to G or C, it is possible that the mutated site return back easily to A or T under the pressure. For example, suppose that a synonymous substitution from A to G takes place at a codon AAA for lysine, changing this codon into AAG for lysine. It has then tendency that a back mutation from AAG to AAA easily takes place by strong mutation pressure toward A+T. Thus, it may not be able to observe any synonymous substitutions even though they are really occurring (Musto et al. 1995).

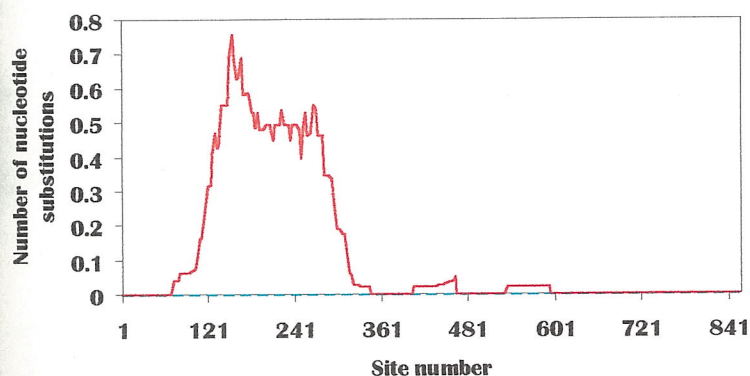


Figure 2. Window analysis of merozoite surface antigen 2 (MSA2) gene of *Plasmodium falciparum*. Numbers of synonymous and nonsynonymous substitutions are plotted along nucleotide sites. Nonsynonymous substitution is indicated by red line. No synonymous substitution was observed in the present pair.

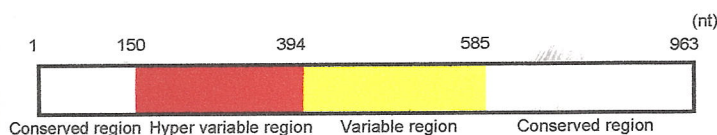


Figure 3. The gene structure of MSA2.

(ii) Case of the major surface protein 1 α of rickettsia

Another example is the major surface protein 1 α (*msp1 α*) genes of rickettsia *Anaplasma marginale*. *A. marginale* can cause anaplasmosis, a hemoparasitic disease in cattle. Msp1 α protein is a subunit of Msp1 protein which is shown to contain a neutralization-sensitive epitope (Oberle et al. 1988). This neutralization-sensitive epitope was mapped to the repeated region where variation in length among clones has been observed. In fact, the *msp1 α* genes code for 78 to 105 kDa proteins depending on the number of repeats (Allred et al. 1990). There were two major regions where d_N is larger than d_S . The first region was from site number 150 to 900 and the second region was from 1060 to 1600 (Figure 4).

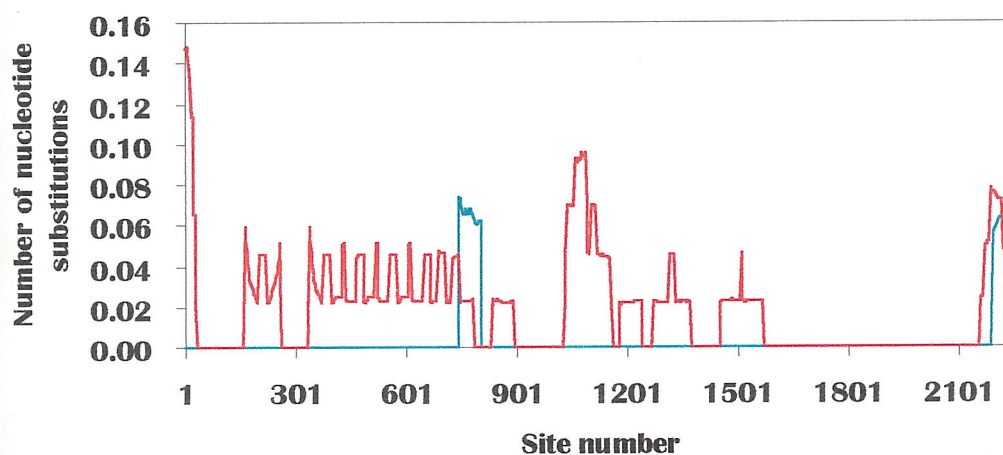


Figure 4. Window analysis of major surface protein 1a (*msp1 α*) gene of *Anaplasma marginale*. Numbers of synonymous and nonsynonymous substitutions are plotted along nucleotide sites. Synonymous and nonsynonymous substitutions are indicated by blue and red lines, respectively.

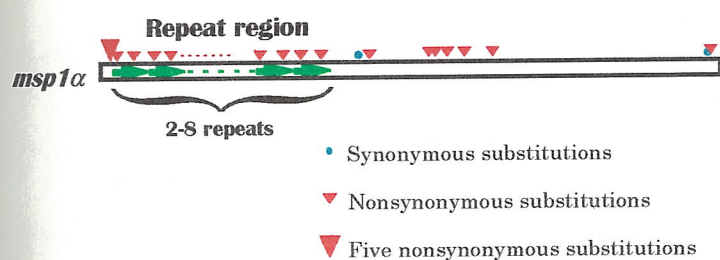


Figure 5. Gene structure of *msp1α*. Repeat units of the gene are indicated by arrows. Nucleotide sites where synonymous and nonsynonymous substitution occurred in two alleles of *msp1α* were indicated by blue and red marks, respectively.

The first region corresponds exactly to the known repeat region where the predicted antigenic epitope is located (Figure 5; Allred et al. 1990). This again suggests that positive selection is operating on the first region within the *msp1α* gene through changes in amino acids to facilitate escape from the host immune system. Although the biological significance of the second region is unknown, it is possible that the second region is also another antigenic epitope which has not been identified yet.

Other genes

The candidate genes can be classified into two groups: surface protein genes of parasites and the others. The two genes described as the examples above are for the surface protein genes. For the remaining candidate genes, I also conducted the window analysis. These results by the window analysis are shown in Figures 6-20. These are an outer membrane protein (*omp*) gene

of *Chlamydia* (Figure 6), an envelope protein (*env*) gene of equine infectious anemia virus (Figure 7), an *E* gene of phages *G4*, a glycoprotein gene *gH* of pseudorabies virus (Figure 8), *FX174* and *S13* (Figure 9), a sigma-1 protein gene of reovirus (Figure 10), and invasion plasmid antigens C and D of *Shigella* (Figures 11 and 12), egg-laying hormone of *Aplysia californica* (Figure 13), egg-laying hormone A of *A. californica* (Figure 14), ATP synthetase F_0 subunit of *Escherichia coli* (Figure 15), neomycin resistance protein of *E. coli* (Figure 16), adhesion molecule gene of *Yersinia* species (Figure 17), prostatic steroid binding protein of rat (Figure 18), neurotoxin genes of snakes (Figure 19), and CDC6 gene of *Saccharomyces cerevisiae* (Figure 20).

According to the figures, we found that there were two classes of gene groups depending on the substitution patterns. One class contains the gene groups in which a large number of nonsynonymous substitutions were observed, compared with a smaller number of synonymous substitutions. The other class contains the gene groups in which a small number of nonsynonymous substitutions occurred with too few number of or even no synonymous substitutions. Let me designate these classes by classes I and II.

Class I includes the cases of *env* gene for equine infectious anemia virus (Figure 7), *E* gene of bacteriophage (Figure 9), *ipaD* gene of *Shigella*

(Figure 12), egg-laying hormone of *A. californica* (Figure 13), *yadA* gene for *Yersinia* adhesion molecule (Figure 17), and prostatic steroid binding protein gene for rat (Figure 18). In these cases, synonymous substitutions observed are not uniformly scattered over nucleotide sites. One possible explanation for this is that strength of functional constraint varies with synonymous sites where nucleotide substitutions may influence the secondary structures of mRNA or any other biological importance at the DNA and/or mRNA level. For instance, it is known that some synonymous sites can be constrained by mRNA secondary structures and the abundance of tRNA (Ikemura 1985). If the secondary structure of mRNA constrains the nucleotide substitution, the pattern of synonymous substitution is expected to become similar to that of nonsynonymous substitution. This is because there is no distinction between synonymous and nonsynonymous substitutions for secondary structures of mRNA. This tendency can be seen for *Aplysia* gene for egg-laying hormone, *Yersinia* gene for adhesion molecule, and rat gene for prostatic steroid binding protein. However, this tendency cannot be seen for bacteriophage E gene and *E. coli* gene for invasion plasmid antigen. Thus, it is unlikely for these genes that the secondary structure of mRNA constrains the synonymous substitutions. Therefore, alternative explanations must be considered for these cases. One possible alternative may be that the site-dependent mutation

rate may cause variation of the number of synonymous substitutions over sites. However, this possibility must be tested by experiments.

Class II contains the gene groups in which a small number of nonsynonymous substitution occurred with few number of synonymous substitutions. In this class, there are *Chlamydia* gene for outer membrane protein (Figure 6), equine infectious anemia virus gene for envelope protein (Figure 7), pseudorabies virus *gH* gene (Figure 8), reovirus sigma-1 protein gene (Figure 10), *Shigella* invasion plasmid antigen gene *ipaC* (Figure 11), *E. coli* gene for ATP synthetase F_0 subunit (Figure 15), *E. coli* gene for neomycin resistance protein (Figure 16), *Yersinia* gene for adhesion molecule (Figure 17), and *Saccharomyces cerevisiae* gene for CDC6 (Figure 20). In general, a small number of synonymous substitutions imply the short divergence time between two genes compared. For example, the equine infectious anemia virus gene for envelope protein shows that only a small number of synonymous substitutions occurred whereas many nonsynonymous substitutions took place (Figure 7). In this case, the divergence times between the genes in this group are possibly short. Therefore, a larger number of nonsynonymous substitutions imply that even for such a short divergence time, strong positive selection may have operated on this gene. Of course, there is a possibility that the number of synonymous

substitution may be due to the pressure of G+C contents or codon biases as seen in the case of *MSA2*. In some other cases where the number of synonymous substitutions is zero, we do not have any ideas for biological explanation. There is a possibility that negative selection is operating against synonymous substitutions, but there is no evidence for it. To conduct more careful examination, more data should be accumulated.

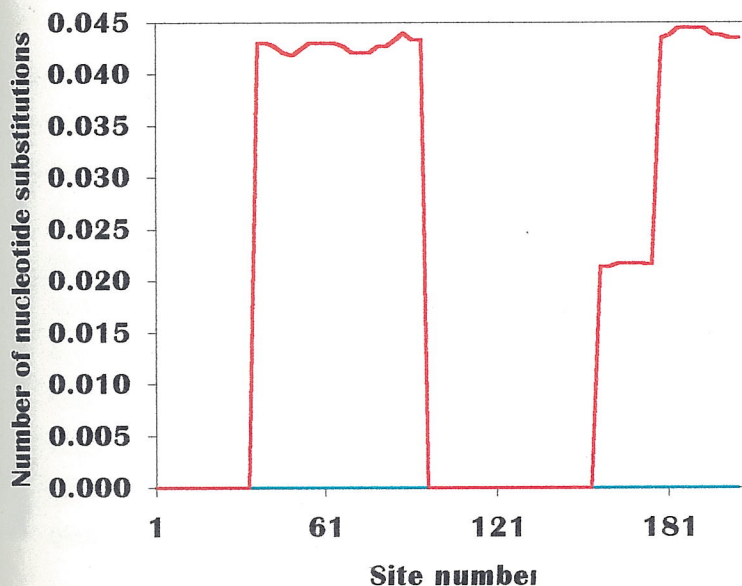


Figure 6. *Chlamydia* outer membrane protein.

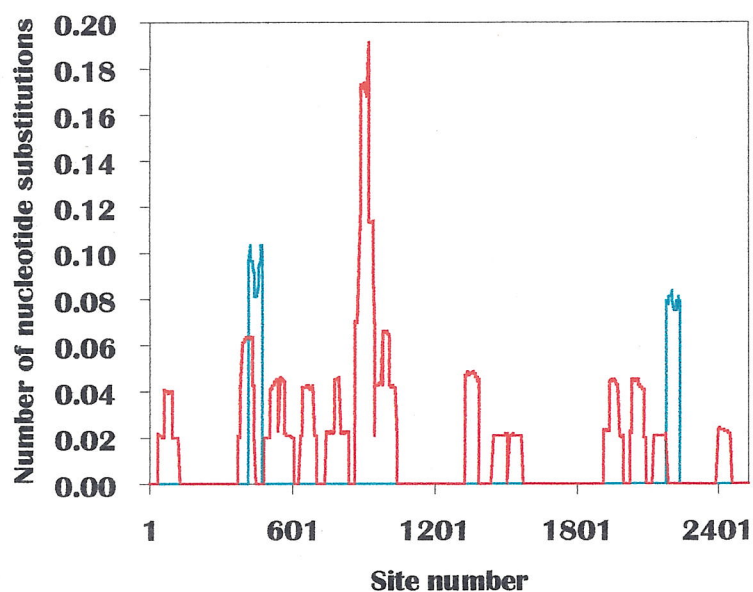


Figure 7. Equine infectious anemia virus envelope protein.

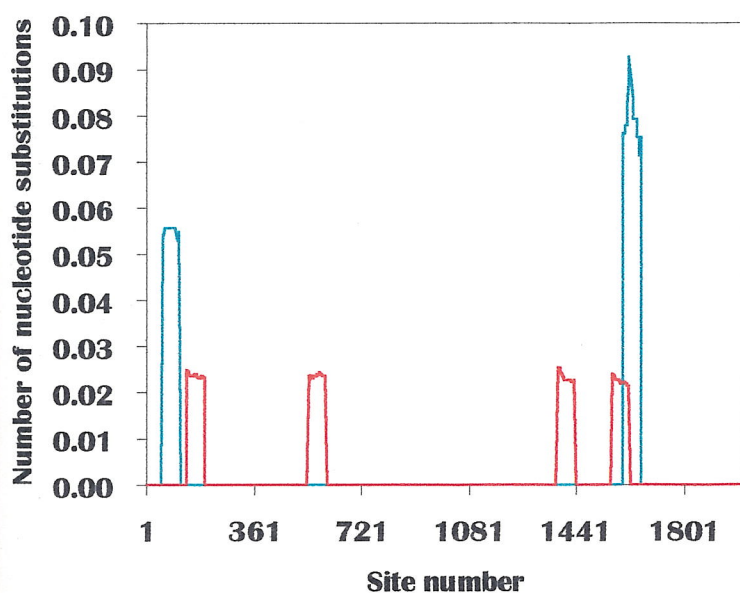


Figure 8. *Pseudorabies virus gH* gene.

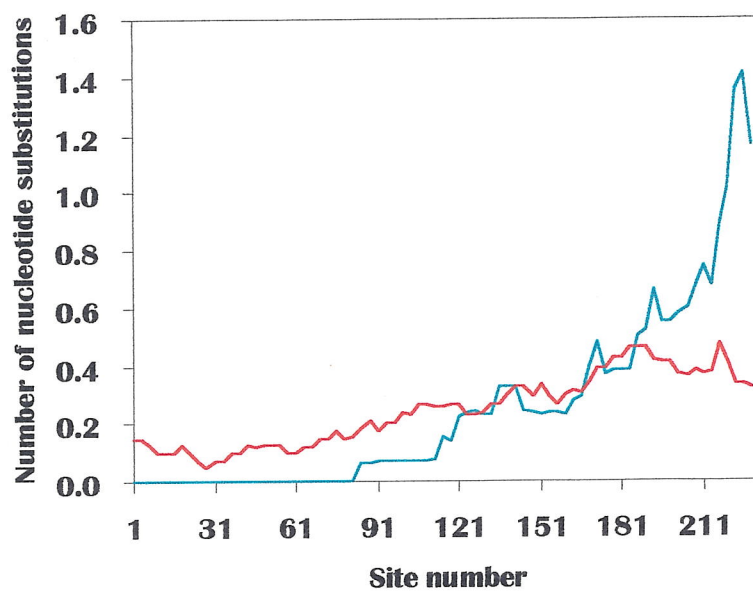


Figure 9. Bacteriophage *E* gene.

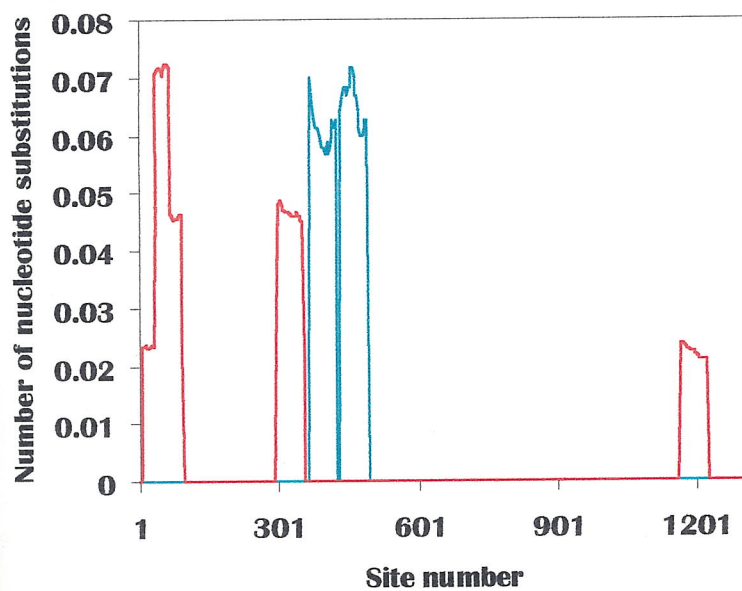


Figure 10. Reovirus sigma-1 protein.

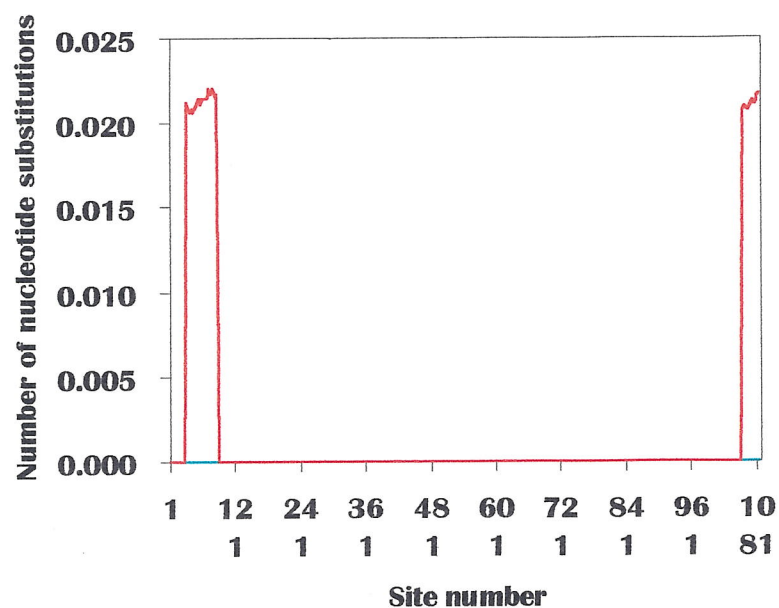


Figure 11. *Shigella* invasion plasmid antigen IpaC.

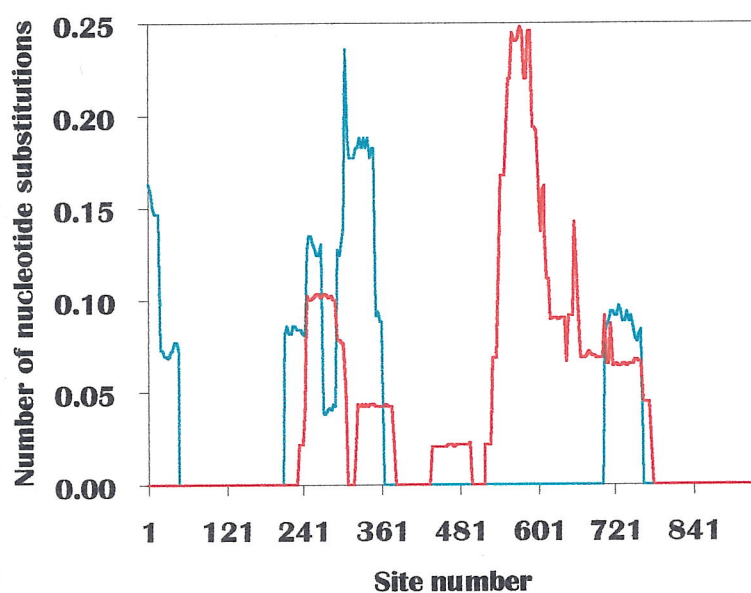


Figure 12. *Shigella* invasion plasmid antigen IpaD

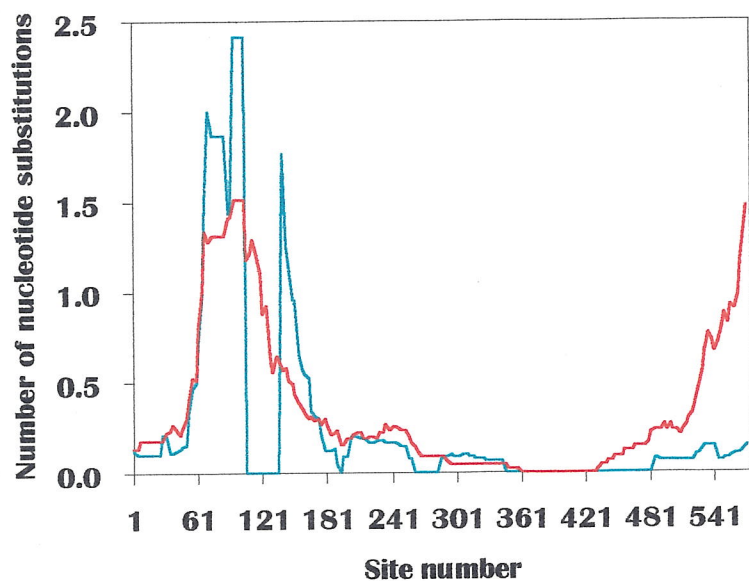


Figure 13. *Aplysia californica* egg-laying hormone.

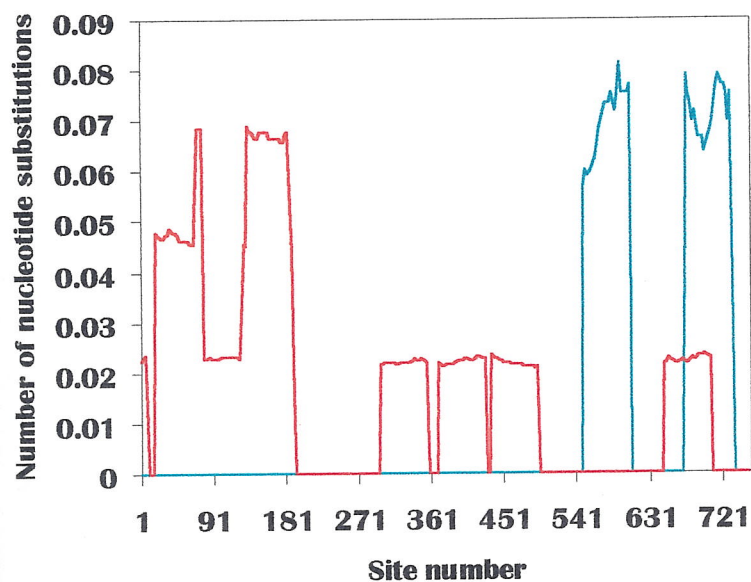


Figure 14. *Aplysia californica* egg-laying hormone A.

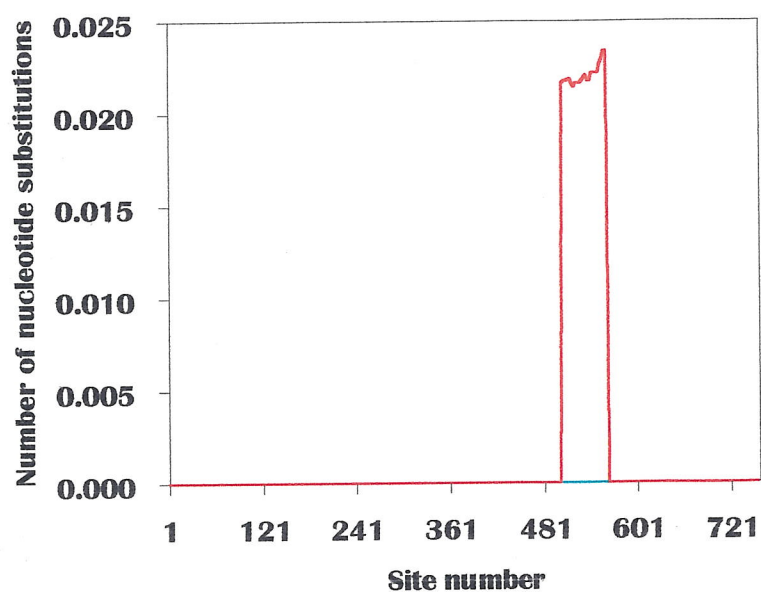


Figure 15. *Escherichia coli* ATP synthase F₀ subunit.

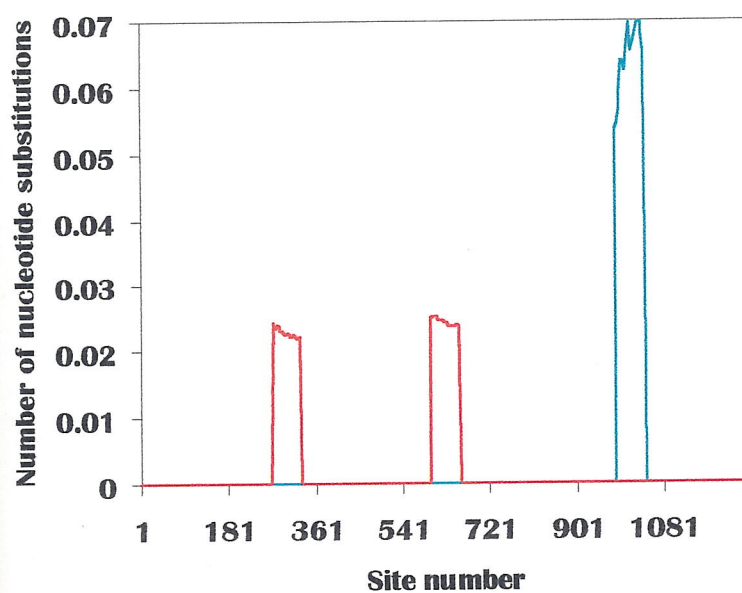


Figure 16. *Escherichia coli* neomycin resistance protein.

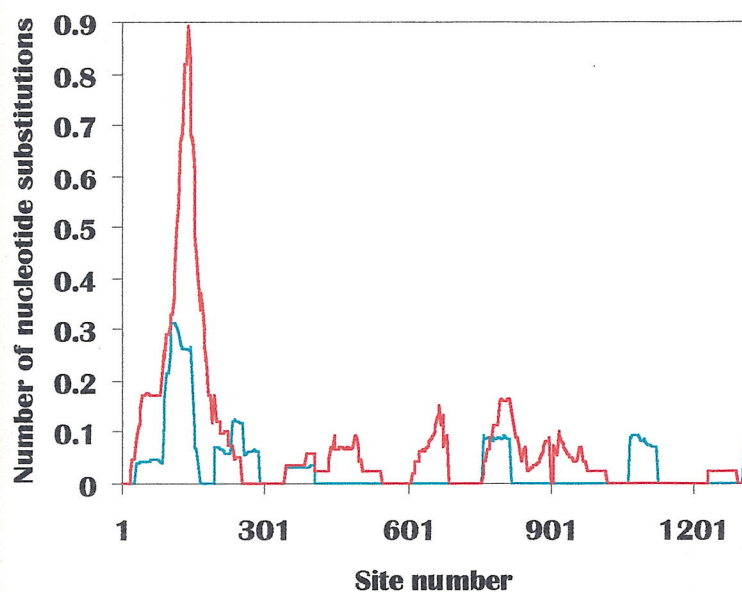


Figure 17. *Yersinia* adhesion molecule.

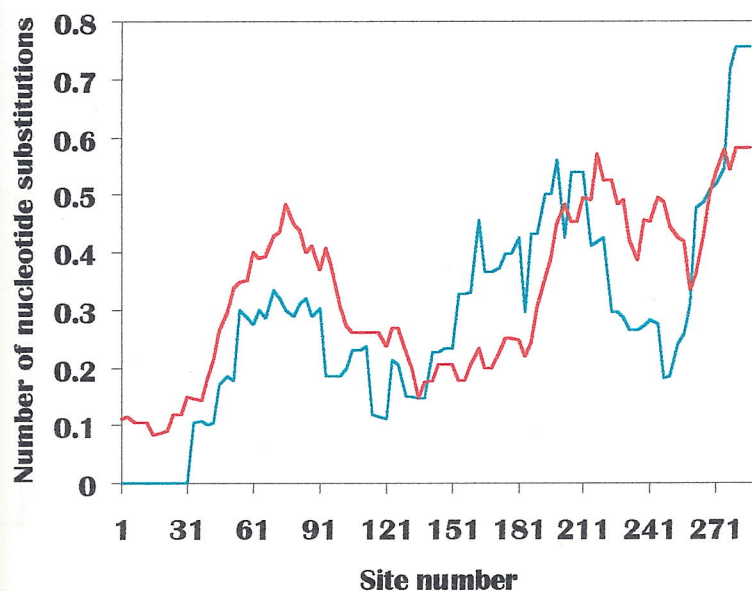


Figure 18. Rat prostatic steroid binding protein.

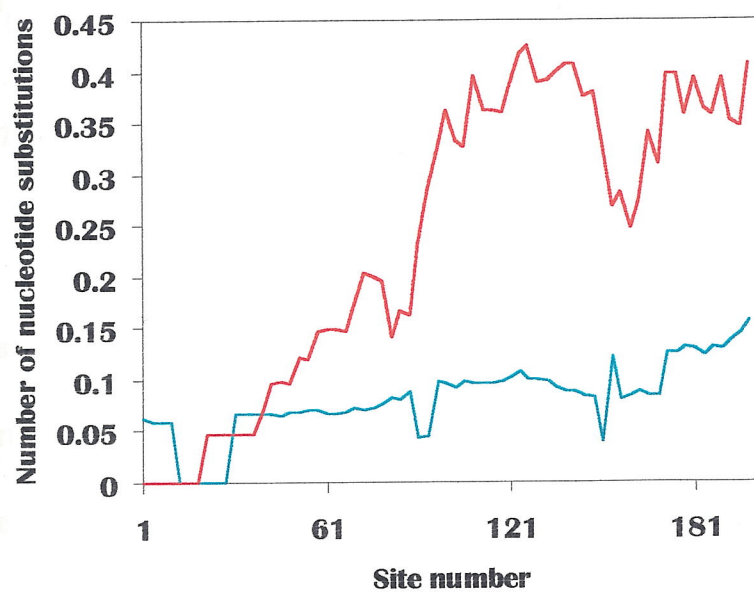


Figure 19. Snake neurotoxins.

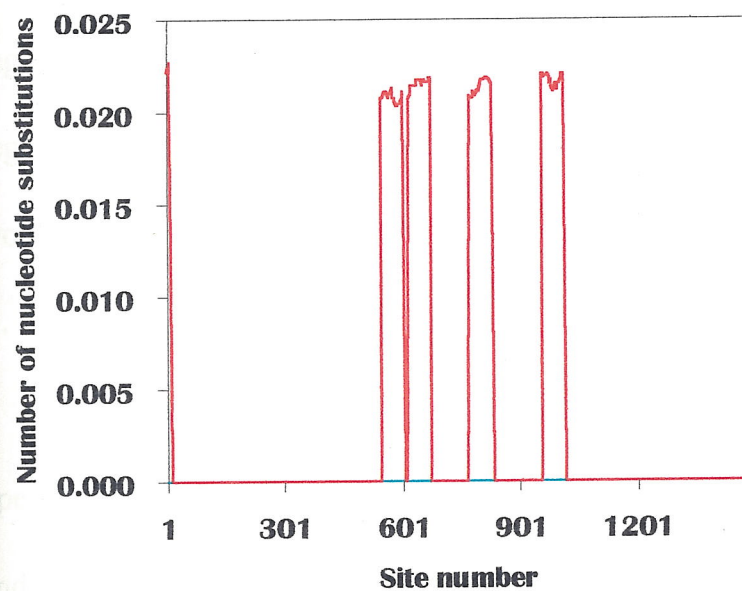


Figure 20. *Saccharomyces cerevisiae* CDC6.

Positive selection seems to operate on only a small proportion of genes

In this study, 17 out of the 3,595 gene groups were found to be candidate genes on which positive selection may operate. This means that for only 0.45% of the gene groups examined, positive selection was considered to be operating. This proportion was very small, as predicted by the neutral theory of molecular evolution (Kimura 1983). However, it should be noted that the candidate genes were identified by estimating d_N and d_S for the entire gene region. For this reason, candidate gene groups in the present study did not contain gene groups in which d_N was not larger than d_S for the entire gene region, but in which d_N was larger than d_S for only certain parts of the gene region. If such groups had been included, the proportion of candidate genes would have been considerably larger. This will be discussed further in PART II.

Hughes and Nei (1988; 1989) showed that positive selection may be operating exclusively at antigen recognition sites on genes in MHC classes I and II by indicating that d_S is significantly higher than d_N at those sites. However, the MHC genes were not included among the candidate genes obtained in the present study. As mentioned above, this is mainly because I considered entire coding region when comparing d_N with d_S for the analysis

and I dealt only with gene groups in which d_N is larger than d_S for the entire coding region. In fact, I found the group in which the MHC genes were identified to have within-gene regions in which d_N is larger than d_S as Hughes and Nei showed (1988), when I conducted the window analysis for a MHC gene groups (Figure 21).

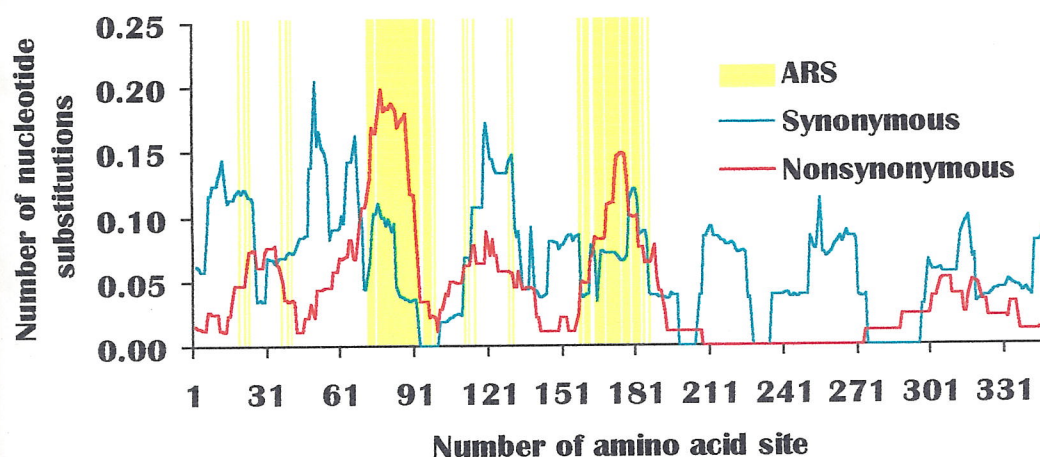


Figure 21. Window analysis of A locus of human class I MHC genes. Numbers of synonymous and nonsynonymous substitutions are shown by red and blue lines, respectively, along the nucleotide sites. Antigen recognition sites are indicated by yellow.

Since Hughes and Nei's studies were reported in 1988, a number of other genes have been investigated. These include stomach lysozyme genes of ruminants (Irwin and Wilson 1990), visual pigment genes (Yokoyama and Yokoyama 1990), homeobox containing genes (Kappen et al. 1989), gamma hemoglobin genes (Fitch et al. 1991), alcohol dehydrogenase genes (Long

and Langley 1993), ion channel genes (Strong et al. 1993), growth hormone genes (Ohta 1993), hemagglutinin gene of human influenza A viruses (Ina and Gojobori 1994), α_1 -antitrypsin genes (Ohta 1995), and abalone sperm lysin genes (Lee et al. 1995). In these studies, the rates of nonsynonymous substitutions were shown, at least, to be accelerated, although they were not necessarily higher than the rates of synonymous substitutions. For this reason, those genes were not always identified as candidate genes in the present study.

Common features among candidate genes

Eight of the 17 candidate genes found through the systematic search analysis were surface proteins of parasites and viruses. These results suggest a possibility that positive selection may be operating on genes for parasite and virus surface proteins, in order to facilitate escape from the immune system of their hosts through frequent nonsynonymous substitutions. To examine this possibility, I conducted a window analysis of two candidate genes, *MSA2* and *mspl α* . For these two gene groups, the window analysis successfully identified the two regions where d_N is larger than d_S within the gene. These regions corresponded very well to the antigenic epitopes of the surface proteins.

Thus, it is possible that the surface proteins of parasites and viruses can be identified as candidate genes by showing that d_N is larger than d_S in their epitope regions.

As for candidate genes other than surface proteins, there may not be any common features among them. More detailed information about the function of these genes is required in order to understand the biological significance of positive selection in those genes.

Summary

It is now known that for most genes so far studied, the rate of synonymous (silent) substitution is higher than that of nonsynonymous (amino acid-altering) substitution. This is consistent with the neutral theory of molecular evolution (Kimura 1968), because synonymous substitutions, by definition, are exempt from functional constraints at least at the amino acid level. However, we cannot necessarily rule out the possibility that positive selection is operating on some other genes. For such genes, the rate of nonsynonymous substitution would be expected to be higher than that of synonymous substitution, because some nonsynonymous substitutions might be favored by natural selection. Thus, for genes on which positive selection may operate, the number of nonsynonymous substitutions would be expected to be larger than that of synonymous substitutions when the nucleotide sequences of genes are compared with each other.

On the basis of this premise, nucleotide substitutions at antigen recognition sites of the major histocompatibility complex (MHC) genes have been shown to be subjected to positive selection (Hughes and Nei 1988, 1989). This kind of study, however, has not been conducted systematically so far. Now, a large amount of DNA sequences are available in the form of database

information. This provides a good opportunity to search for such genes systematically by classifying all the protein-coding sequences in the DNA sequence database into an appropriate number of homologous gene groups.

By building a database of 3,595 groups of homologous sequences from the DNA sequence databases of DDBJ, EMBL and GenBank, I found that 17 gene groups were candidates for genes on which positive selection may operate. Thus, such genes were found to occupy only 0.5% of the vast number of gene groups so far available. Interestingly enough, eight out of the 17 gene groups were surface antigens of parasites or viruses.

I further attempted to identify the within-gene region which may be responsible for positive selection by using the window analysis method. In the case of two parasites genes, merozoite surface antigen 2 (MSA2) of malaria *Plasmodium falciparum* and major surface protein 1 α of rickettsia *Anaplasma marginale*, the predicted region responsible for positive selection was found to include the antigenic epitope regions. I also found other regions which have not been identified as antigenic epitopes. Thus the window analysis method in this study may facilitate to identify unknown antigenic epitopes or other regions which may be responsible to positive selection.

PART II

Positive Selection May Operate on 5% of the Genes Sequenced So Far

Introduction

A gene can evolve faster than the mutation rate if positive selection operates on it. At the nucleotide level, the rate of nonsynonymous substitution is expected to increase relative to that of synonymous substitution when positive selection operates. It follows that the number (d_N) of nonsynonymous substitutions is larger than that (d_S) of synonymous substitutions when the homologous sequences are compared with each other. Using this feature as the criterion, Hughes and Nei (1988, 1989) showed that positive selection operates on the antigen recognition sites (ARS) of the major histocompatibility complex genes.

In PART I, I showed 17 candidate genes on which positive selection may operate by reconstructing 3,595 alignments of homologous coding sequences and examining and selecting cases where d_N is larger than d_S . The MHC genes, however, were not included among the 17 candidate genes. The problem was that d_N was compared with d_S for the entire coding region. It is generally true that when positive selection operates on a gene, the effect is limited to certain within-gene regions rather than the whole coding region where d_N is smaller than d_S . Hughes and Nei were concerned only with collecting the known antigen recognition sites in their study to examine whether or not positive selection operates on. However, identification of the functional region is difficult and the functional regions are still unknown for most of the genes.

In this study, I attempted to find such genes, by focusing on relatively small regions rather than whole coding sequence without knowledge of functional regions or tertiary structures. For this purpose, I developed *the window search method* to search systematically for genes containing the within-gene regions where positive selection might operate. By applying this method to all the alignments of coding sequences which were described in PART I, I obtained 192 candidates.

Materials and methods

Window search method

The window search method was designed to search for genes with regions in which positive selection operates. This method is based on pairwise comparisons of nucleotide sequences within a homologous gene group. For all possible combinations of sequences in each homologous gene group, d_S and d_N are estimated for each window starting from every 10th codon in the pair. The window size used in this study was 20 codons. If $d_N > 2d_S$ for at least one window in a pair, the gene group was regarded as a candidate containing within-gene regions where positive selection may operate (Figure 22). We used $d_N > 2d_S$ rather than $d_N > d_S$ as the criterion because of the standard errors involved and to decrease calculation time. The obtained results were analyzed using the following statistical test.

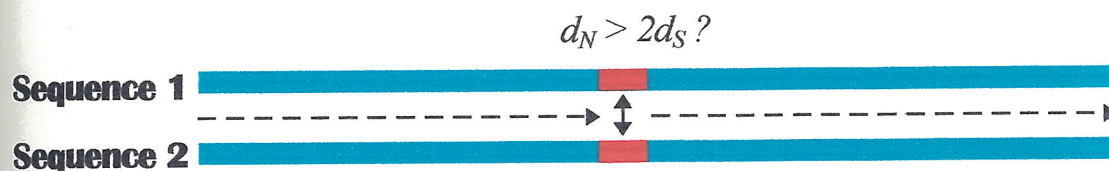


Figure 22. The window search method. The number of nonsynonymous substitutions is compared with that of double of synonymous substitutions for each sliding window.

Statistical Test

The statistical significance of $d_N > d_S$ for candidate genes obtained in the analysis was tested using the following formula:

$$r = \frac{d_N - d_S}{\sqrt{V(d_S) + V(d_N)}}$$

The significance level was regarded to be 1% and 5% if the r values exceeded 2.33 and 1.65, respectively.

Results and Discussion

About 5% of genes may be subject to positive selection

Table 3 show lists of 192 gene groups obtained as candidate genes within which positive selection may operate, using the window search method against the 3,595 coding sequences of homologous sequences which were described in PART I. They thus made up 5.3% of all the gene groups examined. In the list of gene groups, the product of four genes were unidentified.

The characteristics of gene groups

The list of candidate genes consists of subgroups which are distinguished by biological characteristics (Table 2 and Table 3). The largest group included the surface protein genes of parasites and viruses as shown in PART I. In contrast to the result in PART I, however, the surface protein gene group constituted only 15.6% of all candidate genes. This may imply that relatively small regions in the coding sequence are subject to positive selection, especially for genes other than those of parasites and viruses. Various kinds of genes are included in the list (Table 3) in addition to the

surface proteins. These included secretory genes, toxin genes, genes for five of six classes of enzymes in the enzyme nomenclature (Web 1992).

Surprisingly, the total number of enzyme genes was 44, which accounted for 23% of the candidate genes. Genes for transcription factors were also found to be candidate genes, although most of them are viral genes. The group of class I MHC genes was also included as one of the candidate gene groups.

Table 2. Categories of the products of the candidate gene.

Type of gene product	Number of gene groups	%*
Surface protein of parasite and virus	30	15.6
Secretory protein	20	10.4
Transferase (EC2 enzyme)	18	9.4
Hydrolase (EC3 enzyme)	11	5.7
Toxin and virulence protein	10	5.2
Oxidoreductase (EC1 enzyme)	9	4.7
Cell adhesion molecule and cell surface antigen	8	4.2
Transcription regulator	7	3.6
Hormone and cytokine	7	3.6
Hormone and cytokine receptors	6	3.1
Structural protein	4	2.1
Lyase (EC4 enzyme)	4	2.1
Proteinase inhibitor	3	1.6
Ligase (EC6 enzyme)	2	1.0
Antibiotics resistance	2	1.0
Other protein	51	26.6
Total	192	100

*Percentage indicates the proportion of all of the candidates.

Gene structure and positive selection

There seems to be no obvious common features among the different candidate genes contained in the within-gene regions where positive selection may operate. However, the groups of surface protein genes were the largest in number. This implies that positive selection occurs to allow parasites and viruses to escape from their hosts' immune system. Other than the surface proteins, many viral genes were included.

Kinds of candidate genes on which positive selection may operate

In this study, we revealed that 5.3% of all genes examined were possibly affected by positive selection. The list (Table 3) of candidate genes does not include genes for basic metabolic pathways such as those for amino acid and nucleotide synthetic systems, the glycolytic pathway, the Krebs cycle or lipid metabolic systems. Since these systems are indispensable for most organisms and are considered, there may be little room for further refinement of the system and thus positive selection hardly operates. Because most of the candidate genes are not shared over species, they are considered relatively new genes. Among the candidate genes, the surface protein genes

of parasites and viruses were the largest in number. Though these genes may or may not be ancient, there still seems to be a need for continuous change to allow escape from the host defensive system. In other categories of genes, there may be a need for continuous change to alter the function or some other reasons.

Bias of DNA sequence database

There is a possibility that the DNA sequence data in the database is so biased that the results obtained in this study do not constitute a generality. In particular, many of the candidate genes were human, mouse, rat, chicken, virus and parasite genes, and most of them seem related to some kind of disease. To generalize the conclusions, many more sequences from other organisms are needed.

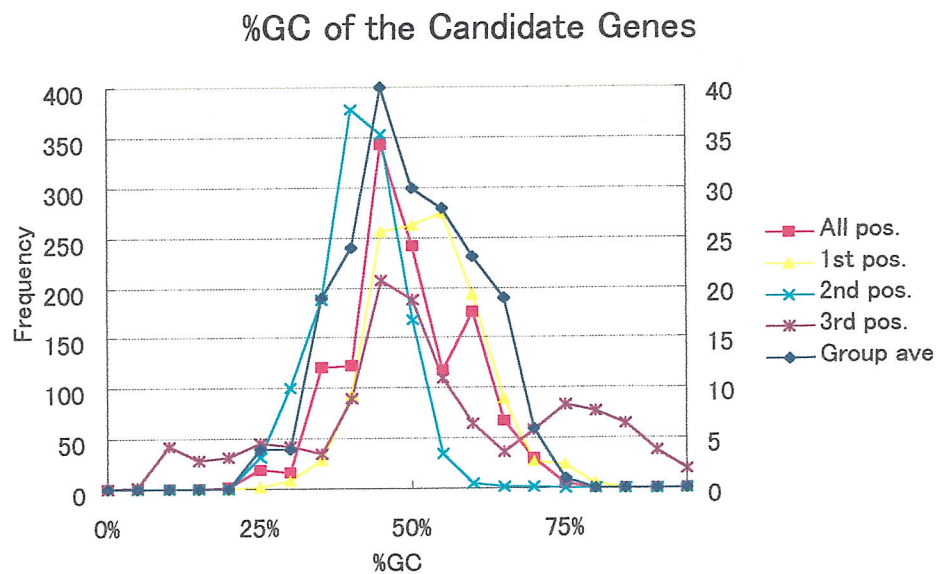


Figure 23. G+C content at each codon position

The frequency of each class of G+C content was plotted for all coding sequences and each codon position. Each class contains frequency of data within 5% width.

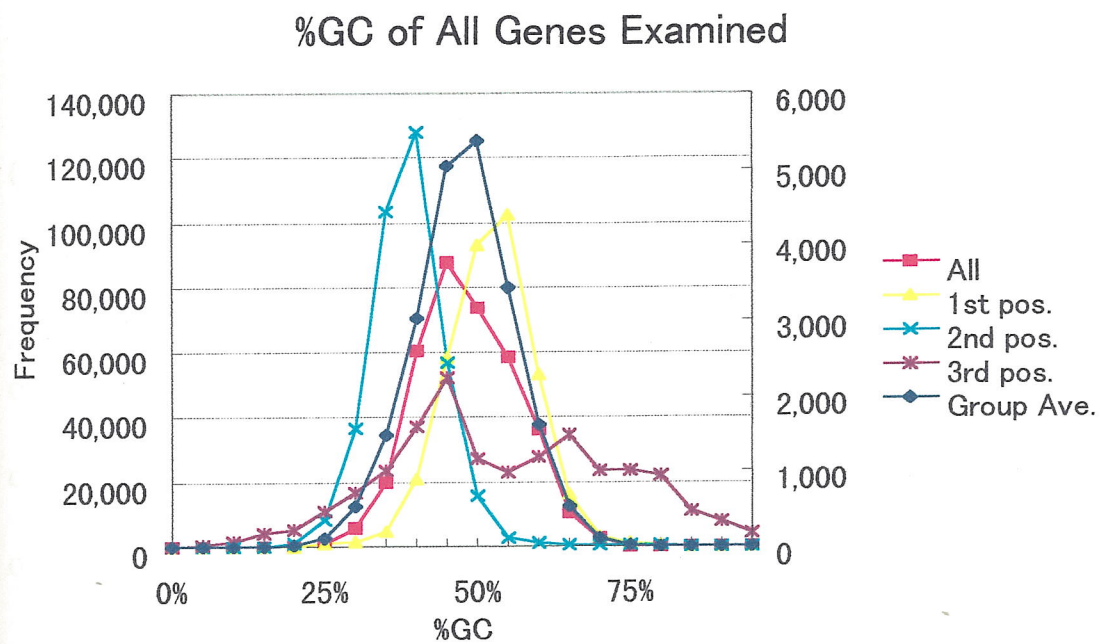


Figure 24 G+C content of all genes examined in the analysis.
Details are the same as in Figure 23.

G+C content of candidate genes

The G+C content of malarial gene *MSA2* for merozoite surface antigen 2 was extremely low especially at the third codon position. Therefore, the reason d_N is larger than d_S could be that synonymous substitutions were suppressed under the low G+C pressure rather than under the pressure of positive selection. To examine whether a biased G+C content is observed in many candidate genes, I checked the G+C content. Figure 23 shows a histogram for G+C content for candidate genes. The distribution of G+C content indicates that extremely low or high G+C content was rather rare. Thus, an extremely biased G+C content does not seem to be common among the candidate genes.

Genes for surface proteins of parasites and viruses

In total, 27 genes for surface proteins of parasites and viruses were obtained as the candidate genes. As the products of these genes are expressed on the surface of membranes, they should be direct targets of immune systems or other defense systems of their hosts. Therefore, it is possible that positive selection operates on the genes in order to allow the gene carriers to escape from the host defense systems.

Genes for secretory proteins

The gene group for secretory proteins contained the second largest in member. A secretory protein usually contains a signal or leader peptide which is not present in mature protein. Within-gene regions on which positive selection possibly operates was not always locate on such regions. Rather, substitutions at the site of a mature peptide may cause functional differentiation, and the nonsynonymous substitution observed in such a region could have altered the function. Secretory proteins interact with their corresponding receptors and the region required for interaction is limited to a small number of sites.

Genes for toxin genes

Twelve genes encoding toxins or virulence determinants, were obtained as candidate genes, and most of these were bacterial genes. In *Yersinia*, for example, a gene was disrupted by a frame shift when it acquired virulence. The gene is expressed on surface membrane of bacteria to attach and to enter its host cell. Therefore, one possible reason for positive selection is that it helps the organism escape from the host immune system.

Cell adhesion molecules and cell surface antigens of non-parasitic genes

Both cell adhesion molecules and cell surface antigens are involved with cell-to-cell interaction and cell recognition. Thus, positive selection may operate on these genes to refine the interaction between these molecules. These changes could occur depending on the molecule so that they may facilitate the differentiation of genes.

Enzyme genes

In total, 44 enzyme genes were included in the list of candidate genes. In particular, 18 genes encode transferase which contains viral DNA polymerase. The active sites of enzymes are usually composed of a small number of amino acid residues and the residues required for interacting with other molecules seemed to be limited in number even though other residues may contributed to its function. Furthermore, the activity or substrate of the enzyme could easily be changed by substitutions at a small number of sites. Thus positive selection would play an important role in enzyme genes.

*Common features among candidate genes containing
within-gene regions where positive selection may operate*

In general, the region of a gene responsible for positive selection seems to interact with small numbers of molecules. For instance, secretory proteins interact with their receptor. Another example is the case of the surface antigen genes of parasites which interact with both the attachment to their target cells and the immune systems of their hosts. The functional constraint on the region where d_N is larger than d_S does not seem very strong. However, these regions would not always be non-functional regions.

Summary

The proportion of genes on which positive selection operates has not been systematically studied, although it is a point of argument among neutralists and selectionists (Kimura 1983). To estimate the proportion of such genes, I conducted a large-scale systematic search for genes on which positive selection may operate. For this purpose, I made 3,595 multiple alignments of homologous coding sequences referring to the homology of the amino acid sequence. Using this multiple alignment database, I systematically performed a large-scale search for selectively advantageous genes by carrying out the window analysis for all the multiple alignments to identify within-gene regions where the number of nonsynonymous substitution is larger than the number of synonymous substitutions. As the result, I obtained 192 of such cases that is considered to be selectively advantageous. They constituted 5.2% of all the homologous gene groups examined. When they were categorized by the characteristics of their products, it was revealed that the category which contained the largest number were the surface protein genes of parasites and viruses, though they constituted only 30 genes among 192 genes.

Table 3. Candidate genes that contain regions possibly subject to positive selection.

Gene product	Representative species	Accession	Type ¹	$d_N > 2d_S$	Significance ²
Surface protein (parasites and viruses)					
1 outer membrane protein (fatty acid transport)	<i>E. coli</i> , <i>Haemophilus influenzae</i>	M60607		181-360, 621-540, 571-720, 751-990, 1081-1230, 1321-1380	301-330*, 331-390**, 631-750**, 871-960*,
2 major pilus subunit	<i>E. coli</i>	M12861	V	1-90, 211-300	
3 outer membrane protein TolC	<i>E. coli</i>	X00016	V	481-660, 721-780, 931-990, 1051-1170, 1291-1380	1111-1200*,
4 cell surface glycoprotein	<i>Halobacterium halobium</i> , <i>H. volcanii</i>	J02767		871-930	
5 outer membrane protein class 1, 3	<i>Neisseria meningitidis</i> , <i>N. gonorrhoeae</i>	X12899	D	1-90, 121-1080	181-270*, 421-510*, 511-570**, 631-690**, 781-840*, 871-930*, 1141-1170*, 1171-1230**, 1231-1251
6 outer membrane protein, opacity-related	<i>Neisseria meningitidis</i> , <i>N. gonorrhoeae</i>	X06445		1-120, 211-300, 601-690	1-60*, 91-150*, 181-240*, 271-330*, 661-720**,
7 outer membrane protein B	<i>Rickettsia rickettsii</i> , <i>R. prowazekii</i>	X16353		421-510, 1591-1650, 2611-2730, 2851-2910, 3151-3240, 3721-3780	481-540*, 2701-2760*, 3211-3270**,
8 surface glycoprotein gp63 protease	<i>Leishmania donovani</i> , <i>L. chagasi</i>	Y00647		31-600, 631-1410, 1471-1560, 1591-1710	121-240*, 301-390*, 541-570*, 571-630**, 691-780*, 841-930*, 1201-1260**, 1351-1440*,
9 merozoite surface antigen 2	<i>Plasmodium falciparum</i>	X53832	V	31-870	151-210*, 271-540**, 541-600*, 631-750*,
10 S-antigen	<i>Plasmodium falciparum</i>	M19155	V	61-210, 751-840, 1171-1230, 1381-1470, 1501-1560, 1591-1710	91-150*, 151-240**, 241-270*, 811-870*, 1231-1290*, 1651-1710*, 1861-1935
11 major merozoite surface antigen gp195	<i>Plasmodium falciparum</i>	M37213	V	91-150, 211-270, 781-960, 1081-1170, 1531-1590, 2251-2340, 2971-3030, 3151-3210, 3301-3390, 3451-3570, 3631-3690, 4411-4470, 4831-4950	271-330**, 841-990*, 2341-2400**, 3211-3270*, 3331-3420*, 3541-3600*,
12 serine-repeat antigen protein	<i>Plasmodium falciparum</i>	J04000	V	1111-1200, 1231-1320, 1861-1920	1291-1350*, 1891-1950**,
13 sialidase (neuraminidase)	<i>Trypanosoma cruzi</i>	M64836	V	241-300, 451-600, 871-990, 1111-1170, 1201-1260, 1291-1350, 1381-1440, 2011-2100, 2131-2190	511-570*, 571-630**, 901-990*, 1141-1200*, 1261-1320*,
14 expression site variant specific antigen	<i>Trypanosoma brucei</i>	M33720	V,D	61-210, 301-540, 841-1050	211-270*,
15 variant surface glycoprotein	<i>Trypanosoma</i>	X01843	V	1321-1410	1381-1440*,

16 envelope protein	<i>feline leukemia virus A,B, murine leukemia virus</i>	M25425		1-1800	61-90*, 91-150**, 181-240*, 271-330*, 361-450*, 691-750*, 871-960*, 1021-1200*, 1411-1500*, 1591-1680*, 2011-2100*, 2221-2286
17 envelope glycoprotein	<i>feline immunodeficiency virus,</i>	X60725		31-270, 361-510, 631-720, 931-990, 1081-1230, 1321-1500, 1621-1860, 1921-2070, 2131-2280, 2371-2430	61-120*, 181-240*, 421-480*, 511-570*, 691-750*, 1111-1200*, 1381-1470*, 1651-1710*, 2221-2310*, 2521-2610
18 latent membrane protein LMP1	<i>Epstein-Barr virus</i>	X58140		1-510, 571-780	1-120*, 391-510*, 631-720*,
19 coat protein	<i>bean leaf roll virus, beet western yellow virus, potato leaf roll luteovirus, potato leaf roll virus, barley yellow dwarf virus</i>	X53865		151-360, 421-480	181-210*, 211-270**, 301-360**,
20 G glycoprotein	<i>bovine syncytial virus, human respiratory syncytial virus</i>	M58307		61-300, 331-390, 631-810	661-720*, 751-840**,
21 hemagglutinin esterase	<i>influenza C virus</i>	M17868	V	1-60, 451-510	
22 surface antigen	<i>ground squirrel hepatitis virus, woodchuck hepatitis virus, human hepatitis B virus, chimpanzee hepatitis B virus</i>	K02715		181-330, 421-840	1-180*, 211-270*, 391-480*,
23 surface protein	<i>duck hepatitis B virus</i>	M60677		1-90, 151-210, 241-450	301-390**,
24 glycoprotein gIV	<i>bovine herpes virus type 1, pseudorabies virus</i>	M59846		151-210	211-270**, 331-390*, 541-600**,
25 glycoprotein D	<i>equine herpes virus type 1</i>	M62923	V	691-840, 871-960, 1081-1170, 1231-1290	
26 envelope	<i>mouse mammary tumor virus</i>	K00556	V		
27 membrane antigen gp350	<i>Epstein-Barr virus</i>	V01555	V	91-150, 361-420, 1531-1590, 1771-1920, 1-90*, 421-480*, 1921-1980*, 2011-2100, 2371-2430, 2461-2610	
28 p96^{gag}	<i>Avian myelocytomatosis virus, Rous sarcoma virus</i>	J02247		61-150, 211-900, 1021-1080, 1291-1350, 421-510*, 601-660*, 1891-1950*, 1411-1560, 1651-1740, 1831-1950, 2641-2700*, 2101-2190, 2281-2400, 2461-2520, 2581-2640, 2701-2760, 2971-3120, 3241-3300, 3451-3570, 4021-4260	
29 Pre-S	<i>Hepatitis B virus</i>	M74498		151-390, 481-870, 901-960	361-450*, 811-870*, 1231-1320*,
30 virulence protein YadI	<i>Yersinia pseudotuberculosis</i>	X13883	V	31-690, 721-840, 931-1170, 1201-1290	151-210*, 271-300*, 301-390**, 601-690*,

	plasmid, <i>Y. enterocolitica</i> plasmid				781-870*, 991-1050*, 1081-1110*, 1111-1170**,
Toxin and virulence protein					
31 virulence determinant E2	<i>Agrobacterium tumefaciens</i> plasmid Ti	X04784	D	121-180	151-210*,
32 virulence determinant D2	<i>Agrobacterium rhizogenes</i> plasmid, <i>A. tumefaciens</i> plasmid Ti	X12867		1-90, 151-240, 811-1170, 1231-1290	901-960*,
33 virulence determinant D3	<i>Agrobacterium rhizogenes</i> plasmid, <i>A. tumefaciens</i> plasmid Ti	X12867		871-990, 1021-1140, 1381-1440, 1591-1650, 1711-1860, 1921-1980	901-960*,
34 pertussis toxin S2, S3 subunit	<i>Bordetella pertussis</i>	M13223	D	481-540	
35 neurotoxin type E	<i>Clostridium botulinum</i> , <i>C. tetani</i> , <i>C. botulinum</i> C phage	X52066		1-120, 211-300, 541-720, 811-930, 991-1050, 1171-1230, 1321-1560, 1651-1710, 1861-1920, 2251-2310, 2491-2610, 2641-2700, 2791-2910, 3031-3150, 3181-3420, 3631-3690	631-690*, 841-900*, 901-960**, 961-1020*, 1231-1320*, 1441-1470*, 1471-1530**, 1531-1590*, 2281-2340*, 2551-2610*, 2731-2790*, 2851-2940*, 3121-3180*, 3241-3300**, 3301-3390*, 3391-3450**, 3451-3480*, 3661-3720**, 3721-3750*, 3781-3840*, 40
36 toxin A, B	<i>Clostridium difficile</i>	X51797	D	121-180, 3181-3270, 6241-6330, 6391-6450, 7111-7170, 7711- 7770, 7981-8040	7441-7500*,
37 phospholipase C	<i>Pseudomonas aeruginosa</i> ,	M13047		571-630, 691-810, 1591-1650	751-810*, 1441-1500*, 1651-1710**, 1831-1890*,
38 hemolysin A	<i>Vibrio cholerae</i>	Y00557	V	31-120, 301-390, 481-540, 961-1050, 1651-1770, 2011-2100	1021-1110**, 1201-1260*, 1711-1770*,
39 eosinophil cationic protein (neurotoxin)	human	X15161	V	1-60, 271-420	361-420**,
Secretory protein					
40 extracellular amylase	<i>A. hydrophila</i> , <i>Xanthomonas campestris</i>	M20401		1261-1320	
41 apolipoprotein A-I	<i>B. taurus</i> , human, baboon, rabbit, <i>M. fascicularis</i> , rat, chicken	M35870		1-90, 151-210, 331-510, 571-690	1-90*, 421-510*, 601-660*, 661-720**, 751-840*,
42 beta-casein	bovine, ovine, human, rabbit, rat	M15132		151-570	241-300*, 331-420*, 571-630*, 721-765

43	beta-lactoglobulin	bovine, ovine, goat, human	X14712		1-150, 241-330, 391-480	61-150*
44	vitamin K-dependent protein S (blood clotting)	bovine, human	M13044		1771-1830	
45	apolipoprotein E	rabbit, human, baboon, macaque, rat	M36603		61-150, 211-480, 511-600, 811-900	301-360*, 871-930*,
46	whey acidic protein	rabbit, rat, mouse	X07943		61-240	391-450
	alpha-2-macroglobulin	human, rat, mouse	M11313		1-60, 91-540, 571-810, 841-930, 1021-1080, 1111-1620, 1651-1740, 1861-1980, 2041-2220, 2281-2370, 2431-2760, 2791-2880, 2941-3030, 3301-3510, 3601-3750, 3781-3870, 3931-3990, 4021-4080, 4111-4260, 4351-4410	181-270*, 271-330**, 331-390*, 511-570*, 901-960*, 1201-1410*, 1921-2010*, 2101-2250**, 2341-2400**, 2731-2790*, 2851-2910*, 3361-3450*, 3691-3780*, 4171-4230*, 4531-4590
47	carcinoembryonic antigen	human, (rat, mouse)	M17303	V	1-120, 181-720, 751-870, 901-1620	1-90**, 91-120*, 211-330**, 331-390*, 391-480**, 481-510*, 511-600**, 601-690*, 691-750**, 841-900**, 901-930*, 961-1050*, 1051-1110**, 1261-1320*, 1471-1530*, 1831-1890*, 1921-1980*,
48	apolipoprotein A-IV	human, mouse, rat	X13629		31-180, 421-510, 571-630, 751-900, 931-1050, 1111-1200	
49	nonspecific cross-reacting antigen	human, mouse, rat	D90278		1-1980	1-90**, 91-120*, 121-180**, 211-240*, 241-330**, 331-390*, 391-480**, 481-600*, 601-690**, 691-780*, 1021-1080*, 1111-1170*, 1261-1320*, 1351-1410*, 1471-1560*,
50	intrinsic factor	human, rat	M63154		601-660, 751-810	
51	glycophorin A	human	M12857	V	1-90	
52	pulmonary surfactant protein	human, mouse, rat	J03553		361-480	421-480*,
53	T-cell activating protein Ly-6Ab	mouse, rat	M74013		121-180, 241-300	
54	mucin Muc-1	mouse	M65132	V	301-420, 1471-1560, 1771-1860	
55	alpha-2mu-globulin	mouse, rat	M27608		121-210, 241-450	301-450*,
56	parotid secretory protein	mouse, rat	X01697		151-270, 361-420, 511-570	181-240*, 241-300**, 301-330*,
57	seminal vesicle secretory protein 5	mouse, rat	X63161		121-210, 271-330	151-210*, 331-390**,
58	caerulein type I, II, IV	<i>Xenopus laevis</i>	X02310	D	91-210, 271-450, 601-690	151-240*, 421-480**,

59 tissue factor	rabbit, human, mouse	M55390	31-120, 751-840
------------------	----------------------	--------	-----------------

Cell surface molecules (non-parasite)

60 CD5 antigen	bovine, human	X53061	31-90	91-150**,
61 osteopontin	bovine, swine, human, mouse, rat	M66236	91-150, 301-360, 751-870	151-210*, 631-690*, 811-870**,
62 monocyte antigen CD14	rabbit, human, mouse	M85233	1-60, 511-600	
63 lymphocyte antigen CD4	chimpanzee, human, rhesus monkey, rat, mouse	M31135	151-390, 421-720, 751-840, 931-990, 1021-1350	301-360*, 961-1020*, 1081-1260*,
64 vesicular cell adhesion molecule 1	human, rat, mouse	M30257	1-60, 1171-1260, 1351-1410	931-990*,
65 erythrocyte membrane protein	human, mouse	M29399	1-90, 841-900	871-930*,
66 killer cell surface molecule	mouse	M77753	V 91-180, 211-450	1-90**, 301-360**, 631-690*, 691-720
67 chicken MHC B-G antigen	chicken	M61860	V 271-360, 541-630, 781-1080	871-930**, 931-960*, 961-1020**, 1021-1140*,

Oxidoreductase (EC1)

68 reductive oxygenase	<i>A. calcoaceticus</i> , <i>Pseudomonas putida</i> plasmid, <i>M. capsulatus</i>	M76990	691-750	
69 nitorogenase subunit K	<i>Azotobacter vinelandii</i> , <i>A. chroococcum</i> , <i>C. pasteurianum</i>	M23528	121-180, 451-540 571-630, 721-840	511-600*, 1111-1170*,
70 glycolate oxidase	<i>Pseudomonas putida</i> , <i>S. oleracea</i>	J05293	601-660	
71 photosynthetic reaction center subunit H	<i>Rhodopseudomonas viridis</i> , <i>R. sphaeroides</i>	X02659	391-450, 601-660	
72 adrenodoxin reductase	bovine, human	M17029	181-240, 331-420, 781-870, 961-1050	991-1080*,
73 photosystem I	barley, rice, spinach	X16092	61-120	
74 cytochrome c oxidase subunit VIc	human, rat	X13238	1-90	31-120*,
75 ribonucleotide reductase large subunit	<i>herpes simplex virus types 1, 2</i>	D10879	151-240, 601-660, 931-990, 1651-1710, 3091-3180	961-1020*,
76 ribonucleotide reductase small subunit	<i>herpes simplex virus types 1, 2</i>	D10879	61-120, 481-570, 661-750	91-150*,

Transferase (EC2)

77	phosphoribulokinase	<i>A. eutrophus</i> , <i>X. flavus</i> , <i>Rhodobacter sphaeroides</i>	M33562	121-240	241-300*,
78	PHA-polymerase	<i>A. eutrophus</i> , <i>P. oleovorans</i>	J05003	541-600, 1081-1170, 4564-1380, 1591-1680	1081-1200*, 1261-1350*, 1651-1710*,
79	DNA methylase	<i>H. gallinarum</i>	D90363	D 811-870	841-900*,
80	B-ketoacyl synthase	<i>Streptomyces curacoi</i> , <i>S. glaucescens</i>	M33704	391-450	421-480*,
81	granaticin polyketide synthase	<i>S. violaceoruber</i> , <i>S. glaucescens</i> , <i>S. curacoi</i>	X16144	91-150, 451-540, 631-690, 1081-1140, 1171-1230	1201-1260*,
82	diacyl glycerol kinase	swine, human	X53256	121-180, 991-1080	
83	starch synthase	barley, wheat, maize	X07931	1-90, 181-270, 391-450, 481-540, 1261-1320, 1351-1410	421-510*, 1411-1470*,
84	α -aminoadipyl-cysteinyI-valine synthetase	<i>A. nidulans</i> , <i>Penicillium chrysogenum</i>	X54853	541-600	
85	mitochondrial AT synthetase subunit 9	<i>A. nidulans</i> mitochondria, <i>N. crassa</i> mitochondria	M30144	91-150	
86	glutathione-S-transferase	maize, <i>N. tabacum</i> , wheat	X04375	1-150, 481-540, 601-660	
87	reverse transcriptase	<i>M. xanthus</i> , <i>Stigmatella aurantiaca</i>	M24392	211-270	241-330*,
88	DNA polymerase	<i>ground squirrel hepatitis virus</i> , <i>woodchuck hepatitis virus</i> , <i>woodchuck hepatitis virus 2</i> , human <i>hepatitis B virus</i> , chimpanzee <i>hepatitis B virus</i>	K02715	1-300, 361-450, 541-1170, 1201-1860, 1921-2160, 2221-2340	1-60*, 121-210**, 631-690**, 691-720*, 721-1170**, 1231-1290*, 1321-1410*, 1501-1530*, 1531-1590**, 1591-1620*, 1621-1680**, 1681-1770*, 1831-1890**, 2281-2340*, 2641-2700
89	DNA polymerase	<i>duck hepatitis B virus</i> , <i>heron</i> <i>hepatitis B virus</i>	M60677	121-540, 661-1380, 1651-1770, 2101-2250, 2311-2370	271-300*, 301-360**, 781-840**, 841-870*, 871-1350**,
90	replicase	<i>barley yellow dwarf virus</i> , <i>potato leaf roll virus</i>	D10206	61-180, 211-300, 601-660, 1321-1440, 1471-1650	181-210*, 211-270**, 301-360**,
91	phosphoprotein	<i>sendai virus</i> , <i>human parainfluenza virus type 1</i>	X17007	1-60, 121-210, 301-570, 721-780, 961-1050, 1141-1230, 1441-1500, 1531-1590	361-480**,

92 protein kinase GCN2	<i>S. cerevisiae</i>	M27082	1171-1260, 1531-1590, 1651-1770, 1951-2040, 2611-2700, 3211-3270	511-570*, 1741-1800*,
93 tyrosine-specific protein kinase ROS1	human, rat,	M34353	3151-3210	

Hydrolase (EC3)

94 Exoglucanase	<i>Cellulomonas fimi</i> , <i>Streptomyces lividans</i>	M15824	391-450	
95 Glutamate dehydrogenase	<i>C. difficile</i> , <i>P. asaccharolyticus</i>	M65250	601-660	
96 nylon degrading enzyme (EII)	Flavobacterium plasmid	X02864	91-210, 601-660	1141-1200*,
97 parathion hydrolase	Flavobacterium, <i>P. diminuta</i> plasmid	M29593	361-420	421-480*,
98 coagulase	<i>Staphylococcus aureus</i>	X17679	931-1050, 1141-1200, 1321-1380	1021-1080*,
99 angiotensin-converting enzyme	rabbit, human, mouse	J05041	1-60, 361-450, 781-870, 1081-1170, 1201-1260, 1381-1500, 1561-1650, 1681-1740, 1771-1920, 2011-2070, 2101-2160, 3391-3450, 3721-3780	1831-1920*, 2071-2130*,
100 AMP deaminase	<i>S. cerevisiae</i>	M30449	61-120	
X 101 sperm lysin	<i>Haliotis rufescens</i> , <i>H. corrugata</i>	M34388	211-330	
102 co-beta glucosidase	human, rat	J03077	871-930	901-960*,
103 nucleoside triphosphate phosphohydrolase	<i>Choristoneura biennis</i> <i>entomopoxvirus</i> , <i>vaccinia virus</i>	M60400	841-900	
104 alkaline exonuclease	<i>herpes simplex virus types 1, 2</i>	K02022	31-90, 841-900	901-960*,

Transcriptional regulator

105 transcriptional activator pdr4	<i>S. cerevisiae</i>	X53830	V 871-930, 1711-1800	
106 MASH	rat	X53725	D 181-240	
107 ssDNA binding protein	<i>human Adenovirus type 4, type 7</i>	M12407	1-60	61-120*,
108 polymerase-associated protein	<i>canine distemper virus</i> , <i>measles virus</i>	M32418	61-540, 631-1200, 1231-1290, 1381-1470	181-270*, 301-360*, 481-510*, 511-570**, 721-780*, 841-990*,
109 transcriptional activator Syn	<i>herpes simplex virus types 1, 2</i>	M16610	211-270, 931-990	
110 regulatory protein E2	<i>human papillomavirus types 1,</i>	K02718	631-720, 841-930	721-780*, 901-960*,

	16,18, 35, 57				
111 b protein	<i>Ustilago maydis</i>	M58553	V	1-180, 211-330, 391-540, 571-720, 781-840, 871-960	1-120*, 151-210*, 241-360*, 481-570*, 631-690*, 691-750**, 1111-1170*,
112 syn (transcriptional activator)	<i>Herpes simplex virus types 1, 2</i> <i>Varicella-Zoster virus</i> , <i>Equine herpesvirus 1</i>	M16610		211-270, 931-990	

Hormone and cytokine

113 egg-laying hormone	<i>Aplysia californica</i>	J01017	V	181-330	511-660**,
114 FMRF amid polypeptide	<i>D. melanogaster</i> , <i>D. virilis</i>	J03232		61-150, 361-450, 601-870, 961-1020	1021-1080*,
115 endothelin-1	bovine, pig, rat, human	X52740		1-90, 361-450	1-90**, 601-675
116 interleukin-1beta	bovine, ovine, human	M37211		1-570	241-270*, 271-330**, 451-510*,
117 preproenkephalin	swine, human	V01362		361-420	
118 interleukin 6	<i>Sus scrofa</i> , human, rat, mouse	M80258		31-120, 511-600	
119 kidney epidermal growth factor	human, mouse, rat	X04571		181-240, 301-360, 961-1020, 1111-1170, 1591-1650, 1951- 2010, 2101-2160, 3421-3480	121-180*, 3361-3420*,
120 preproendothelin	bovine, pig, rat, human	X52740		43-651	1-90**, 601-675
121					

Hormone and cytokine receptors

122 interleukin-2 receptor	bovine, ovine, mouse	M20818		91-180, 331-390, 481-630, 751-810	541-600*, 601-660**,
123 poly-Ig receptor	rabbit, rat	K01291		61-120, 1561-1620, 2101-2160	1621-1680*, 2161-2220*,
124 macrophage colony stimulating factor receptor	human, rat	M37435		241-360, 541-600, 991-1170, 1201-1410	1051-1110*, 1381-1440*,
125 granulocyte colony-stimulating factor receptor	human, mouse	M59818		1291-1350, 2341-2430, 2461-2520	1-60*,
126 vitamin D receptor	human, rat	J03258		451-510	
127 steroid hormone receptor	<i>D. melanogaster</i>	X15586	V	511-570, 721-780, 1381-1440, 1561-1620, 1711-1770, 3601-3690, 4111-4200	1411-1470*, 4141-4200**,

Structural protein

128 flagellin A1	<i>H. halobium</i>	M19883	D	181-330, 421-510	
129 major fimbrial subunit	<i>Haemophilus influenzae</i>	X16991	V	241-300, 481-600	541-630*,
130 M-protein	<i>Streptococcus pyogenes</i>	M19031	V	121-300, 391-450, 481-570, 601-990, 1231-1320, 1441-1620	1-90*, 181-210*, 211-360**, 541-600*, 661-720*, 1711-1770*,
131 Tenascin	swine, chicken, mouse, human	X61599		571-720, 901-990, 1081-1170, 1291-1410, 1591-1680, 2701-2790, 3031-3120, 3181-3270, 3721-3810, 3841-3900, 4111-4260, 4381-4440, 4831-5250	1351-1440*, 3781-3870*, 4441-4500*,
132 capsid	<i>herpes simplex virus type 1, equine herpesvirus type 1</i>	D10879		31-180, 901-990	961-1020*,
133 viral protein 1 (capsid)	<i>mouse minute virus, feline panleukopenia virus, canine parvovirus, mink enteritis virus, porcine parvovirus</i>	M12032		31-180, 241-330, 361-420, 511-690, 721-870, 901-990, 1231-1290, 1381-1890	751-900*, 1471-1560*, 1651-1680*, 1681-1740**, 2251-2340
134 capsid VP1	<i>murine minute virus, canine parvovirus, mink enteritis virus, feline panleukopenia virus, porcine parvovirus</i>	M12032	V	31-180, 241-330, 361-420, 511-690, 721-870, 901-990, 1231-1290, 1381-1890	751-810*, 1591-1650**, 2161-2250
135 capsid VP2	<i>murine minute virus, feline panleukopenia virus, mink enteritis virus, raccoon parvovirus, canine parvovirus, porcine parvovirus</i>	M12032		61-240, 421-600, 631,780, 811-900, 1081-1200, 1291-1800	751-810*, 1591-1650**, 2161-2250
136 structural polypeptide (capsid and envelope)	<i>rubella virus</i>	M15240		181-240, 781-840, 871-930, 1051-1110, 1171-1320, 1741-1800, 1921-1980, 2701-2760	
137 VPg	<i>beet western yellows virus, potato leaf roll luteovirus, barley yellow dwarf virus</i>	X13062		1-90, 121-420	1-90*, 211-330*,

Lyase (EC4)					
138 aldolase	<i>Rhodobacter sphaeroides</i>	M64624	D	901-960	241-300*,
139 pectin lyase	<i>Aspergillus niger</i>	X60724	D	721-780	
140 carbonic anhydrase	<i>C. reinhardtii</i>	D90206	D	1-60, 181-300, 361-510, 631-750, 841-900, 991-1050	1111-1170*,
141 adenylate cyclase	<i>Trypanosoma equiperdum</i> , <i>T. brucei</i> kinetoplast	X59385		61-240, 571-690, 991-1080, 1261-1440	

Proteinase inhibitors					
142 uteroferrin-associated protein	swine, ovine, rat	M30315		241-330, 541-630	301-360**, 601-660**, 1321-1350
143 proteinase inhibitor I	potato, tomato	M17108		31-210	151-210*, 361-378
144 cysteine proteinase inhibitor(oryzacystatin)	rice	J03469	D	61-120	

Ligase (EC6)					
145 carboxylase (biotin, pyruvate, propionyl-CoA)	<i>E. coli</i> , <i>S. cerevisiae</i> , rat	M80458		61-120, 901-990, 1081-1170, 1471-1560, 1621-1710, 2941-3000, 3271-3330	3331-3390*,
146 nitrogen regulation protein	<i>E. coli</i> , <i>Klebsiella pneumoniae</i> , <i>Rhodobacter capsulata</i> , <i>B. parasponia</i>	X05173		151-210, 301-360	

Antibiotics resistance					
147 rRNA N-methyltransferase	<i>Arthrobacter</i> , <i>Streptomyces thermotolerans</i> , <i>S. fradiae</i> , <i>S. lividans</i>	M11276		451-510 811-870	
148 aminoglycoside phosphotransferase	<i>Streptomyces griseus</i> plasmid, <i>S. fradiae</i> plasmid, <i>S. ribosidifucus</i> plasmid,	M28829		121-210, 451-540, 631-690	

Others						
149 arylphorin	<i>Calliphora vicina</i>	M76479	V	421-480, 1321-1410		
150 chorion protein s16	<i>D. melanogaster</i> , <i>D. subobscura</i> , <i>D. grimshawi</i>	X16715		31-90		
151 serendipity locus gene beta	<i>D. melanogaster</i>	X03121	D	31-90, 631-690, 751-840		661-720*,
152 bactericidal permeability increasing protein	bovine, human	X52563		271-330, 1081-1140		
153 dopamine transporter	bovine, rat, human	M80234		31-150, 181-240, 391-480, 601-660, 1591-1680, 2041-2100		661-720*,
154 cell lysis	<i>phage G4, S13, phi-X174</i>	J02454		1-150		1-60**, 61-120*, 121-180**, 181-210*,
155 gene 4.3	<i>phage T3, T7</i>	X17255		121-240		181-240**,
156 vicilin storage protein	<i>Canavalia gladiata</i> , <i>Vicia faba</i> , <i>Phaseolus vulgaris</i> , <i>P. glauca</i>	X06733		1-90, 301-360, 511-600, 631-810, 961-1110, 1171-1260, 1321-1410, 1561-1620, 1651-1710		
157 Opaque protein	maize	X16618	V	1171-1290		1291-1350*,
158 S1 protein	<i>Petunia inflata</i> , <i>Nicotiana glauca</i>	M67990		241-300, 331-390, 511-570		241-330*,
159 patatin	potato	X01125	D	1-240, 301-450, 481-960		1-120**, 121-150*, 331-450*, 601-660**, 661-720*, 721-780**, 781-870*, 871-930**, 931-990*, 1051-1140*, 631-690*, 1441-1500*, 1681-1740*,
160 agglutinin	castor bean	M12089	V	181-300, 571-630, 1291-1350		
161 prolamin	rice	X14392	V	1-270		61-120*, 121-180**, 181-210*, 241-300*,
162 nodulin-20	soybean	X05020	D	511-600		571-630*,
163 HMW glutenin subunit 1By9	wheat	X61026	V	31-150, 181-270, 331-1050, 1081-1200, 1261-1620, 1651-1710, 1771-2160		241-300**, 391-480*, 631-690*, 781-840*, 901-990*, 1051-1110*, 1141-1200*, 1291-1350*, 2251-2310*, 2371-2490*, 2581-2640*, 2641-2700**,
164 ribonucleoprotein Ro	human	J04137	V	661-750, 871-960, 1501-1560		
165 lymphocyte activation marker Blast-1	human, mouse, rat	X06341		1-120, 151-240, 271-330, 571-630		1-60*, 751-765
166 rythroid protein 4.1	human, <i>X. laevis</i>	J03796		301-390, 631-750, 2311-2400		691-750*,

167 fibrinogen alpha-chain	human, rat	J00127		511-600, 991-1050, 1351-1410	
168 interferon-gamma induced protein	human	M63838	V	301-450, 601-720	421-480*, 661-720**, 721-750*, 2161-2220*,
169 T cell differentiation marker	mouse, rat	X52991		121-420, 571-660	181-270*, 841-900
170 intra-cellular sub-maxillary gland glycoprotein	mouse, rat	X00349		91-480	151-300*, 421-510*,
171 nuclear antigen	<i>Epstein-Barr virus</i>	V01555	V	391-450, 691-780, 811-870, 901-990, 1051-1140, 1591-1650, 1831-1920, 1981-2100, 2131-2190, 2761-2880, 3061-3120	91-150*
172 nef protein	<i>human immunodeficiency virus type 1</i>	M15654	V	1-750	1-120**, 121-210*, 271-312
173 rex protein	<i>human T-cell leukemia virus type 1, 2</i>	M10060		61-240, 271-360, 421-480	91-180**, 181-210*, 301-330*, 331-390**,
174 inclusion body protein	<i>cauliflower mosaic virus,</i>	J02046	V	1-570, 601-750, 1111-1230, 1261-1530	151-240*, 301-390*, 391-480**, 691-750*, 1471-1560*,
175 phosphoprotein P	<i>human parainfluenza virus type 3,</i> <i>bovine parainfluenza virus type 3</i>	D00048		241-330, 361-630, 901-1020, 1381-1470	1-60**, 61-90*, 361-420**, 481-570**, 571-630*,
176 antifreeze protein	<i>A. lupus, Macrozoarces americanus</i>	M22125		31-150	241-315
177 cytochrome c ₁	<i>Rhodopseudomonas sphaeroides,</i> <i>Rhodobacter capsulata, R. rubrum</i>	X03476		151-210, 511-630	631-690*,
178 cytochrome c ₂	<i>P. denitrificans,</i> <i>Rhodopseudomonas capsulata,</i> <i>R. sphaeroides,</i> <i>Rhodospirillum rubrum</i>	M27304		91-180, 211-270	
179 cytochrome P-450	<i>Saccharopolyspora erythraea</i>	M83110		91-150, 211-270, 391-600, 631-750	181-240*, 451-480*, 481-540**, 541-570*, 571-630**, 661-720*, 721-780**, 781-810*, 841-900*, 961-1020*, 1171-1305
180 prostatic steroid binding protein subunit C1,C2	rat	V01255	D	1-90, 121-180	1-90**, 91-120*,
181 nitrogen fixation protein	<i>A. vinelandii, K. pneumoniae,</i> <i>Clostridium pasteurianum</i>	M20568		271-360, 421-510, 721-840	1081-1140*,

182 fructose enzyme II (FruA)	<i>R. capsulatus, X campestris</i>	M62785	811-900	871-930*,
183 sensor protein	<i>B. paraptussis,</i> <i>B. bronchiseptica, B. pertussis</i>	X52948	31-180, 1171-1230, 1561-1620, 1981-2070, 2191-2250, 2341-2400, 2521-2580, 2821-2940, 2971-3120, 3151-3360, 3511-3660	2881-2940**, 2941-2970*, 3031-3090*,
184 embryo globulin (storage protein)	barley, wheat, maize	M64372	1-240, 691-780, 1291-1470, 1621-1680, 1711-1860	31-90*, 1771-1830*,
185 invasion plasmid antigen IpaD	<i>Shigella flexneri,</i> <i>Shigella dysenteriae</i>	J04117	271-330, 451-750	
186 polyketide synthase CurC, TcmJ	<i>Streptomyces curacoi,</i> <i>S. coelicolar, S. glaucescens</i>	M33704	391-450	421-480*,
187 granaticin polyketide synthase TcmL, CurB	<i>S. violaceoruber,</i> <i>Streptomyces glaucescens,</i> <i>S. coelicolar, S. curacoi</i>	X16144	91-150, 451-540, 631-690, 1081-1140, 1171-1230	1201-1260*,
188 TcmI	<i>Streptomyces glaucescens,</i> <i>S. coelicolar</i>	M80674	151-210	421-480*,
189 p19 protein	<i>Artichoke mottled crinkle virus,</i> <i>Tomato bushy stunt virus,</i> <i>Cymbidium ringspot tombusvirus,</i> <i>Cucumber necrosis virus</i>	X51456	1-60, 91-240	151-180*, 181-240**, 331-450*, 511-564
190 69kD ORF	<i>Turnip yellow mosaic virus</i>	J04373	31-240, 271-390, 451-540, 571-780, 871-1260, 1381-1470, 1561-1710	91-150*, 331-420*, 631-690*, 991-1140*, 1141-1200**, 1201-1290*, 1471-1530*, 1621-1710*,
191 unknown protein	<i>M. musculus</i>	M38248	1-60, 91-180, 211-390, 481-630	331-420*,
192 deduced polypeptide	<i>Tomato black ring virus</i>	D00142	V 151-210, 811-870, 931-990	181-240*,

The type field indicates known allelic variations (V) and duplicated genes (D). ²Significance levels at 5% and 1% are indicated by * and **, respectively.

References

PART I

1. AHO, A. and ULLMAN, J. 1977. **Principles of Compiler Design**. Addison Wesley.
2. ALLRED, D. R., McGUIRE, T. C., PALMER, G. H., LEIB, S. R., HARKINS, T. M., McELWAIN, T. F., and BARBET, A. F.. 1990. Molecular basis for surface antigen size polymorphisms and conservation of a neutralization-sensitive epitope in *Anaplasma marginale*. **Proceedings of National Academy of Science, USA** 87: 3220-3224.
3. ALTSCHUL, S. F., STEPHEN, F., GISH, W., MILLER, W., MYERS, E. W., and LIPMAN, D. J.. 1990. Basic local alignment search tool. **Journal of Molecular Biology** 215: 403-10.
4. BJORKMAN, P. J., SAPER, M. A., SAMRAOUI, B., BENNETT, W. S., STROMINGER, J. L., and WILEY, D. C. 1987. Structure of the human class I histocompatibility antigen, HLA-A2. **Nature** 329:506-512.
5. BJORKMAN, P. J., SAPER, M. A., SAMRAOUI, B., BENNETT, W. S., STROMINGER, J. L., and WILEY, D. C. 1987. The foreign antigen binding site and T cell recognition regions of class I histocompatibility antigens. **Nature** 329:512-518.
6. DANSE, J. M. and GARNIER, J. M. 1990 A cDNA sequence encoding a neurotoxin-homolog from *Bungarus multicinctus*. **Nucleic Acids Research** 18:1045.
7. DANSE, J. M. and GARNIER, J. M. 1990. cDNA deduced amino-acid sequences of two novel k-neurotoxins from *Bungarus multicinctus*. **Nucleic Acids Research** 18:1050 .
8. ELLIOTT, J. F., ALBRECHT, G. R., GILLADOGA, A., HANDUNNETTI, S. M., NEEQUAYE, J., LALLINGER, G., MINJAS, J. N., and HOWARD, R. J.. 1990. Genes for *Plasmodium falciparum* surface antigens cloned by expression in COS cells. **Proceedings of National Academy of Science, USA** 87: 6363-6367.
9. FENTON, B., CLARK, J. T., ANJAM KHAN, C. M., ROBINSON, J. V., WALLIKER, D., RIDLEY, R., SCAIFE, J. G., and McBRIDE, J. S. 1991. Structural and antigenic polymorphism of the 35- to 48-kilodalton merozoite surface antigen (MSA-2) of the malaria parasite *Plasmodium falciparum*. **Molecular and Cellular**

- Biology 11:** 963-971.
10. FITCH, D. H. A., BAILEY, W. J., TAGLE, D. A., GOODMAN, M., SIEU, L., and SLIGHTOM, J. L.. 1991. Duplication of the γ -globin gene mediated by repetitive L1 LINE sequences in an ancestor of simian primates. **Proceedings of National Academy of Science, USA** **88**: 7396-7400.
 11. FUSE, N., TSUCHIYA, T., NONOMURA, Y., MENEZ, A., and TAMIYA, T. 1990. Structure of the snake short-chain neurotoxin, erabutoxin c precursor gene. **European Journal of Biochemistry** **193**:629-633.
 12. GOJOBORI, T., MORIYAMA, E. N., IKEO, K., INA, Y., IMANISHI, T., TATENO, Y., and FUJITSU LIMITED. 1993. **SODHO Ver.2.0. CD-ROM**. Fujitsu Limited, Tokyo.
 13. HUGHES A. L. 1991. Circumsporozoite protein genes of malaria parasites (*Plasmodium* spp.): Evidence for positive selection on immunogenic regions. **Genetics** **127**: 345-353.
 14. HUGHES A. L. 1992. Positive selection and interallelic recombination at the merozoite surface antigen-1 (*MSA-1*) locus of *Plasmodium falciparum*. **Molecular Biology and Evolution** **9**, 381-393.
 15. HUGHES A. L. and NEI, M. 1988. Nucleotide substitution at major histocompatibility complex loci reveals overdominant selection. **Nature** **335**:167-170.
 16. HUGHES A. L. and NEI, M. 1989 Nucleotide substitution at major histocompatibility complex class II loci: evidence for overdominant selection. **Proceedings of National Academy of Science, USA** **86**:958-962.
 17. IKEMURA T. 1985. Codon usage and tRNA content in unicellular and multicellular organisms. **Molecular Biology and Evolution** **2**:13-34.
 18. INA, Y. and T. GOJOBORI. 1994. Statistical analysis of nucleotide sequences of the hemagglutinin gene of human influenza A viruses. **Proceedings of National Academy of Science, USA** **91**:8388-8392.
 19. IRWIN, D. M., and A. C. WILSON. 1990. Concerted evolution of ruminant stomach lysozymes. **Journal of Biology Chemistry** **265**:4944-4952.
 20. IRWIN, D. M., R. T. WHITE, and A. C. WILSON. 1993. Characterization of the cow stomach lysozyme genes: repetitive DNA and concerted evolution. **Journal of Molecular Evolution** **37**:355-366.
 21. KAPPEN, C., K. SCHUGHART, and F. H. RUDDLE. 1989. Two steps in the evolution of antennapedia-class vertebrate homeobox genes. **Proceedings of National Academy of Science, USA** **86**:5459-5463.
 22. KEMP, D. J., R. L. COPPEL, and R. F. ANDERS. 1987. **Annual Review of**

- Microbiology** 41:181-208.
- 23.KIMURA, M. 1968. Evolutionary rate at the molecular level. **Nature** 217:624-626.
 - 24.KIMURA, M. 1983. **The neutral theory of molecular evolution**. Cambridge University Press, London.
 - 25.KING, J. L., and T. H. JUKES. 1969. Non-Darwinian evolution. **Science** 164:788-798.
 - 26.KORNEGAY, J. R., T. D. KOCHER, L. A. WILLIAMS, and A. C. WILSON. 1993. Pathways of lysozyme evolution inferred from the sequence of cytochrome b in birds. **Journal of Molecular Biology and Evolution** 37:367-379.
 - 27.LEE, Y.-H., T. OTA, and V. D. VACQUIER. 1995. Positive selection is a general phenomenon in the evolution of abalone sperm lysin. **Molecular Biology and Evolution** 12: 231-238.
 - 28.LONG, M., and C. H. LANGLEY. 1993. Natural selection and the origin of *jingwei*, a chimeric processed functional gene in *Drosophila*. **Science** 260:91-95.
 - 29.MARSHALL, V. M., R. L. COPPEL, R. K. MARTIN, A. M. J. ODUOLA, R. F. ANDERS, and D. J. KEMP. 1991. A *Plasmodium falciparum* MSA-2 gene apparently generated by intragenic recombination between the two allelic families. **Molecular Biochemistry and Parasitology** 45:349-351.
 - 30.MARUYAMA, T and M. NEI. 1981. **Genetics** 98:441-459.
 - 31.MUSTO, H., H. RODRIGUEZ-MASEDA and G. BERNARDI. 1995. Compositional properties of nuclear genes from *Plasmodium falciparum*. **Gene** 152:127-132.
 - 32.NACHMAN, M. W., S. N. BOYER, and C. F. AQUADRO. 1994. Nonneutral evolution at the mitochondrial NADH dehydrogenase subunit 3 gene in mice. **Proceedings of National Academy of Science, USA** 91:6364-6368.
 - 33.NEEDLEMAN, S. B. and C. D. WUNSCH. 1970. A general method applicable to the search for similarities in the amino acid sequences of two proteins. **Journal of Molecular Biology** 48:443-453.
 - 34.NEI, M. 1975. **Molecular Population Genetics and Evolution**. North-Holland, Amsterdam.
 - 35.NEI, M. and T. GOJOBORI. 1986. Simple methods for estimating the numbers of synonymous and nonsynonymous nucleotide substitutions. **Molecular Biology Evolution** 3:418-426.
 - 36.OBARA, K., N. FUSE, T. TSUCHIYA, Y. NONOMURA, A. MENEZ, and T. TAMIYA. 1989. Sequence analysis of cDNA encoding a erabutoxin b from the sea-snake *Laticauda semifasciata*. **Nucleic Acids Research** 17:10490.
 - 37.OBERLE, S. M., G. H. PALMER, A. F. BARBET, and T. C. McGUIRE. 1988.

- Molecular size variations in an immunoprotective protein complex among isolates of *Anaplasma marginale*. **Infectious Immunology** **56**: 1567-1573.
38. OHTA, T. 1993. Pattern of nucleotide substitutions in growth hormone-prolactin gene family: a paradigm for evolution by gene duplication. **Genetics** **134**:1271-1276.
 39. OHTA, T. 1991. Multigene families and the evolution of complexity. **Journal of Molecular Biology and Evolution**. **33**:34-41.
 40. OHTA, T. 1994. On hypervariability at the reactive center of proteolytic enzymes and their inhibitors. **Journal of Molecular Biology and Evolution** **39**:614-619.
 41. ROWAN, E., F. DUCANCEL, Y. DOLJANSKY, A. HARVEY, A. MENEZ, and J. C. BOULAIN. 1990. Nucleotide sequence encoding a 'synergistic-like' protein from the venom glands of *Dendroaspis angusticeps*. **Nucleic Acids Research** **18**:1639.
 42. SMYTHE, J. A., R. L. COPPEL, K. P. DAY, R. K. MARTIN, A. M. J. ODUOLA, D. J. KEMP, and R. F. ANDERS. 1991. Structural diversity in the Plasmodium falciparum merozoite surface antigen 2. **Proceedings of National Academy of Science, USA** **88**: 1751-1755.
 43. SMYTHE, J. A., R. L. COPPEL, G. V. BROWN, R. RAMASAMY, D. J. KEMP, and R. F. ANDERS. 1988. Identification of two integral membrane proteins of Plasmodium falciparum. **Proceedings of National Academy of Science, USA**. **85**:5195-5199.
 44. SMYTHE, J. A., M. G. PETERSON, R. L. COPPEL, A. J. SAUL, D. J. KEMP, and R. F. ANDERS. 1990. Structural diversity in the 45-kilodalton merozoite surface antigen of Plasmodium falciparum. **Molecular Biochemistry Parasitology**. **39**:227-234.
 45. STRONG, M., K. G. CHANDY, and G. A. GUTMAN, 1993. Molecular evolution of voltage-sensitive ion channel genes: on the origin of electrical excitability. **Molecular Biology Evolution**. **10**:221-242.
 46. TAMIYA, T., A. LAMOUDROUX, J. F. JULIEN, B. GRIMA, J. MALLET, P. FROMAGEOT, and A. MENEZ. 1985. Cloning and sequence analysis of the cDNA encoding a snake neurotoxin precursor. **Biochemie** **67**:185-189.
 47. WILSON, A. C. 1975. Evolutionary importance of gene regulation. Stadler Symposium. **7**:117-134. University of Missouri, Columbia, Mo.
 48. YOKOYAMA, S. and R. YOKOYAMA. 1990. **Molecular evolution of visual pigment genes and other G-protein-coupled genes**. pp. 307-322 in **Population Biology of Genes and Molecules**, edited by T. OHTA and K. AOKI. Baifukan, Tokyo.

PART II

1. Kimura, M. 1993. **The neutral theory of molecular evolution.** Pp. 149-193. Cambridge University Press, New York.
2. Webb, E., C. 1992. **Enzyme Nomenclature. Recommendations (1992) of the Nomenclature Committee of the International Union of Biochemistry and Molecular Biology.** Academic Press, Inc. California.

APPENDIX

COMPUTER PROGRAMS

bst, bstn (Perl): perform bootstrap resampling of amino acid (bst) or nucleotide (bstn) sequence and summarize the result.

```
#!/usr/bin/perl -w
# bst? - get bootstrap probability.
# this script uses bstrap (oden).
# T. Endo 1995.3.9
$COMPILE = "%M% %I% %H%";

##### files and settings.

$REPEAT = 1000;
$TMP = "$0.tmp$$";
$TREESUFFIX = '.nj';
$T = &DataType;
$Ncolumn = 4; $SUFFIX = ".bst$T"; # for nj tree.
$Ncolumn = 3; $SUFFIX = ".bsu$T"; # for upg tree.

sub Init;
sub MakePrimaryTree;
sub DoBootStrap;
sub PrintResults;
sub handler;

##### check argument.

while ( ( $_ = $ARGV[0] ), /^-/ ) {
    shift;
    ($REPEAT = $1), last if /^-r(\d+)/;
    ($REPEAT = shift), last if /^-r/;
}

die "Set the number of replication." if $REPEAT == 0;

print STDERR "Number of repeat = $REPEAT.\n";

Init;

for $FILE (@ARGV){
    print(STDERR "Can't find $FILE: $!\n"), next unless -f $FILE;
    ($TREEFILE = $FILE) =~ s/(.*)?$/TREESUFFIX/;
    ($BSTFILE = $FILE) =~ s/(.*)?$/SUFFIX/;

    MakePrimaryTree;
    DoBootStrap;
    PrintResults( $REPEAT, '' );
}
```

```

}
unlink $TMP;

sub DataType {
    $0 =~ /[ansg]$/ and return $&;
    print STDERR "Cannot recognize the data type.";
    exit (-1);
}

sub Init {
    ##### method to calculate the distance.

    ($T eq 'a') && ( # for amino acid.
        $METHOD = '-m1' # Zuckerkandl and Pauling's method (1965).  $d = -\ln(1-p)$ .
        # $METHOD = '-m2' # Kimura's method (1983).  $d = -\ln(1 - p - 0.2 * p ** 2)$ .
    );

    ($T eq 'n') && ( # for DNA.
        $METHOD = '-m6' # 6-parameter method (Gojobori et al. 1982).
        # $METHOD = '-m1' # 1-parameter method (Jukes and Cantor 1969).
        # $METHOD = '-m2' # 2-parameter method (Kimura 1980).
        # $METHOD = '-m3' # 3-parameter method (Kimura 1981).
        # $METHOD = '-m4' # 4-parameter method (Takahata and Kimura 1981).
        # $METHOD = '-m5' # Tajima and Nei's method (1984).
        # $METHOD = '-m7' # 1-parameter gamma distance method (Jin and Nei 1990).
        # $METHOD = '-m8' # 2-parameter gamma distance method (Jin and Nei 1990).
    );

    ($T eq 's') && ( # for synonymous and nonsynonymous substitutions.
        $METHOD = '' # synonymous substitutions.
        # $METHOD = '-n' # nonsynonymous substitutions.
        # $METHOD = '-g' # gamma distance of nonsynonymous substitutions.
    );

    ##### methods.

    $DIST = "dist$T -x";
    $DMAT = "dmat$T $METHOD";
    $TREE = "treenj -g";
    $OPTION = "";
    $BSTSITE = "bstrapsite $OPTION";
    $BST = "bootstrap -r";

    # check arg and environment.
    (@ARGV == 0) && die "Usage: $0 alignment_file\n";
    die "Run 'oden' first." unless $ENV{'ODEN'};

    # set signal handler.

    $SIG{'INT'} = 'handler';
    $SIG{'QUIT'} = 'handler';
    $SIG{'SEGV'} = 'handler';
    $SIG{'BUS'} = 'handler';
}

sub handler {
    my ( $sig ) = @_ ;
    print STDERR "Process killed by SIG$sig\n";
    close TMP;
}

```

```

PrintResults( $rep - 1, $sig );
unlink $TMP;
exit(1);
}

sub MakePrimaryTree {
    # make tree file.
    print STDERR "Constructing primary tree - ";
    @TREE = `cat $FILE | $DIST | $DMAT | $TREE`;
    open TREE, ">$TREEFILE" or die "Can't open $TREEFILE : $!";
    print TREE @TREE;
    close TREE;
    chop( @TREE );

    @OTUNAME = ();
    @OTU = ();
    @BRANCH = ();
    undef(%BRANCH);
    foreach (@TREE){
        /^OTU (\d+):\t(\w+)/ and do {
            $OTUNAME[ $1 ] = $2;
            $OTU[ $1 ] = $1;
            next;
        };
        /^\\D/ && next;

        @_ = split;
        next unless @_ == $Ncolumn;

        ( $otu1, $otu2 ) = @_[ 0, 2 ];
        $OTU[ $otu1 ] = $OTU[ $otu2 ] = $otu1; # make node info.

        @list = (); @inv = ();
        for (1 - $#OTUNAME){
            $OTU[ $ _ ] = $otu1 if $OTU[ $ _ ] == $otu2;
            push( @list, $ _ ) if $OTU[ $ _ ] == $otu1;
            push( @inv, $ _ ) if $OTU[ $ _ ] != $otu1;
        }
        $list = '<' . join(' ', @list) . '>';
        $inv = '<' . join(' ', @inv) . '>';
        push(@BRANCH, $list);
        $BRANCH{$list} = $list . $inv;
    }
    print STDERR "Done.\n";
}

sub DoBootStrap {
    # bootstrap seeds.
    print STDERR "Creating bootstrap seeds - ";
    system "$BSTSITE $OPTION -r $REPEAT $FILE > $TMP";
    print STDERR "Done.\n";

    # perform bootstrap and count the results.
    undef %NUM;
    $time0 = $time1 = time; # to predict the time of finish.

    for $rep ( 1 - $REPEAT ){

        $BSTCMD = "$BST $rep $TMP < $FILE | $DIST | $DMAT | $TREE";
        open(BSTCMD, "$BSTCMD|") || die "Can't exec '$BSTCMD':$!";

        @otu = ( 0 - $#OTUNAME );
    }
}

```

```

while ( <BSTCMD> ) {
    /^d/ or next;

    @_ = split;
    next unless scalar( @_ ) == $Ncolumn;
    ($otu1, $otu2) = @_[0,2];
    $otu[$otu1] = $otu[$otu2] = $otu1;
    @list = (); @inv = ();
    for (1 - $#OTUNAME){
        ($otu[$_] == $otu2) && ($otu[$_] = $otu1);
        ($otu[$_] == $otu1) && push(@list, $_)
                               || push(@inv, $_);
    }
    $list = join(' ',@list);
    $inv = join(' ',@inv);
    ++$NUM{$otu} if ($otu = grep(/<$list>|<$inv>/, @BRANCH));
}

# time count.
$sec = int ( ( time - $time0 ) / $rep * ( $REPEAT - $rep ));
if ($sec>60){
    $day = int( $sec / 86400 ); $sec -= $day * 86400;
    $hour = int( $sec / 3600 ); $sec -= $hour * 3600;
    $min = int( $sec / 60 ); $sec -= $min * 60;
    printf STDERR "$FILE: # $rep/$REPEAT (%ds) - ".
        "(Elapsed %d', remains %dd %02d:%02d')\n",
        time - $timel,
        ( time - $time0 ) / 60, $day, $hour, $min + 1;
    $timel = time;
}else{
    printf STDERR "$FILE: Repeat # $rep - (remains %02d sec.)\n", $sec;
}

}

}

sub PrintResults {
    my( $REPEAT, $sig ) = @_;

    # print results.
    print STDERR "Printing bootstrap result into $BSTFILE - ";
    open(BSTFILE, ">$BSTFILE") || die "Can't open $BSTFILE: $!";
    select BSTFILE;
    if ( $sig ne '' ) {
        print ">Interrupted at $REPEAT\n";
    }
    @last = ();
    foreach ( @TREE ){
        $_ eq '' or /^D/ and print( "$_\n" ), next;

        @_ = split;
        if ( @_ == $Ncolumn ) {
            printf "$_\t%5.3f\n", $NUM{ shift @BRANCH } / $REPEAT;
        } else {
            push @last, $_;
        }
    }
    print "$last[1]\t$last[2]\t1.000\n$last[3]\t1.000\n";
    close BSTFILE;
    printf STDERR "Done.\n";

    # finish message.
    print STDERR "Bootstrap finished for $FILE.\n";
}

```

```
if ( @OTU <= 30 ){  
  for ( sort keys %BRANCH ){  
    printf STDERR "$BRANCH{ $_ } %5.1f %%\n",  
      $NUM{ $_ } * 100 / $REPEAT;  
  }  
}  
print STDERR "Bootstrap result is in $BSTFILE.\n";  
}
```


cds1.y, cds2.y (yacc): to extract coding sequences from DDBJ/GenBank formatted database WITH (cds1) or WITHOUT (cds2) length checking.

```

/*
   cds1.y - to extract coding sequence from GenBank/DDBJ format text file.
   T. Endo 1994.1.5, 1994.2.22, 1994.2.24 1994.3.15 3.28
*/
%{
char*CREDIT=
    "cds1.y - to extract cDNA from sequence file.\n\tT. Endo 1994.3.28\n";
%}

%token A C J D N
%%
e: C e ')' { $$=(int)reverse($2); }
  | J l ')' { $$=$2; }
  | r      { $$=(int)grow($1); }
  ;
l: l ',' e { $$=(int)$3; }
  | e      { $$=(int)$1; }
  ;
r: A N D N { $$=(int)newbranch($1,$2,$4); }
  | N D N   { $$=(int)newbranch("", $1,$3); }
  | A N     { $$=(int)newbranch($1,$2,$2); }
  | N       { $$=(int)newbranch("", $1,$1); }
  ;
%%
char *debug=0;
int TOTAL=0;
int TOTAL_CDS=0;
int PRINTED_CDS=0;
int ERRORS=0;

#define Case break;case

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define Nlocus 11
#define Nacc 7
#define Ndiv 4
#define Ndef 1000
#define MAXSEQ 315000

#define CDS_SEPERATOR '\001'
char CDS_SEP_STR[]={CDS_SEPERATOR,0};

char *sgcstr[]={
    "SGC0 - the standard genetic code",
    "SGC1 - the vertebrate mitochondrial code",
    "SGC2 - the yeast mitochondrial code",
    "SGC3 - the mold mitochondrial and mycoplasma code",
    "SGC4 - the invertebrate mitochondrial code",
    "SGC5 - the ciliate macronuclear and dasycladacean nuclear code",
    "SGC6 - the protozoan mitochondrial code",
    "SGC7 - the plant mitochondrial code - due to RNA editing",
    "SGC8 - the echinoderm mitochondrial code",
    "SGC9 - euplotid macronuclear code"
};

enum {SGC0=0,SGC1,SGC2,SGC3,SGC4,SGC5,SGC6,SGC7,SGC8,SGC9};

```

```

typedef struct {
    char acc[Nacc];
    int s,e;
    char *seq;
}reg;

typedef struct bt{
    struct bt *left,*right;
    reg *data;
}btree;

typedef struct{
    char locus[Nlocus];
    int size;
    int sgc;
    char acc[Nacc];
    char div[Ndiv];
    char def[Ndef];
    int ncds;
    char cds[BUFSIZ*4];
    int nseq;
    char seq[MAXSEQ];
} entry;
entry ent;

FILE*f;
char b[BUFSIZ];
char *yystr,*parsestr;
btree *root;

/* functions for yacc parser */
yylex(){
    extern char *yystr;
    extern yylval;

    while (isspace(*yystr)||*yystr=='#')++yystr;
    if (isdigit(*yystr)){
        int n=*yystr-'0';
        while (isdigit(++yystr)) n=n*10+*yystr-'0';
        yylval=n;
        return N;
    }else if (isupper(*yystr)){
        static char acc[Nacc];
        strncpy(acc,yystr,6);
        acc[6]=0;
        yystr+=7;
        yylval=(int)acc;
        return A;
    }else if (*yystr=='.' && yystr[1]==''){
        yystr+=2;
        return D;
    }else if (*yystr==',' || *yystr==''){
        return *yystr++;
    }else if (strncmp(yystr,"complement(",11)==0){
        yystr+=11;
        return C;
    }else if (strncmp(yystr,"join(",5)==0){
        yystr+=5;
        return J;
    }else if (strncmp(yystr,"order(",6)==0){
        yystr+=6;
        return J;
    }else if (isspace(*yystr)){

```

```

        return 0;
    }else
        return -1;
}

yyerror(str)char *str;{
    fprintf(stderr,"%s:%s in %s.\n>%s",str,parsestr,ent.locus,yyst);
    ERRORS++;
    return;
}

btree *
reverse(t)btree *t;{
    int x;
    btree*u;

    x=t->data->s;
    t->data->s=t->data->e;
    t->data->e=x;

    u=t->left;
    t->left=t->right;
    t->right=u;

    if (t->left) reverse(t->left);
    if (t->right) reverse(t->right);
}

char *
getseq(acc)char*acc;{
    static char b[BUFSIZ];
    char *seq;
    int n=0,i,len=0;
    long filepos=ftell(f);

    rewind(f);
    while (fgets(b,BUFSIZ+1,f)){
        if (strncmp(b,"LOCUS",5)==0){
            for (i=22; !isdigit(b[i]); ++i);
            len=b[i]-'0';
            while (isdigit(b[++i])) len=10*len+b[i]-'0';
        }else if (strncmp(b,"ACCESSION",9)==0
            && strncmp(acc,b+12,Nacc-1)==0)
            break;
    }

    seq=(char*)malloc(sizeof(char)*(len+1));

    while (fgets(b,BUFSIZ+1,f)){
        if (strncmp(b,"ORIGIN",5)==0) break;
        else if (strncmp(b,"//",2)==0){
            if (debug) fprintf(stderr,"No entry: %s\n",acc);
            ERRORS++;
            free(seq);
            fseek(f,filepos,0);
            return 0;
        }
    }
    while (fgets(b,BUFSIZ+1,f)){
        if (strncmp(b,"//",2)==0) break;
        for (i=10; b[i]; ++i)
            if (isalpha(b[i])) seq[n++]=b[i];
    }
}

```

```

    seq[n]=0;

    fseek(f,filepos,0);
    return seq;
}

btree *
newbranch(a,b,c) char*a; int b,c; {
    reg*r=(reg*)malloc(sizeof(reg));
    btree*t=(btree*)malloc(sizeof(btree));

    if (r&&t){
        /* newbranch body */
        strcpy(r->acc,a);
        r->s=b;
        r->e=c;
        if (strcmp(a,"")==0)
            r->seq=ent.seq;
        else{
            r->seq=getseq(a);
        }

        /* binary tree */
        t->data=r;
        t->left=t->right=0;
        return t;
    }else{
        fprintf(stderr,"memory overflow");
        exit(-4);
    }
}

btree *
grow(b)btree*b; {
    if (root==0)
        root=b;
    else {
        btree*a=root;
        while (a->right) a=a->right;
        a->right=b;
    }
    return b;
}

char
complement(x) char x; {
    static char base[] = "ABCDGHKMNRSTUVWXYabcdghkmnrstuvwxy";
    static char cbase[] = "TVGHCDMKNYWAABSXRtvghcdmknywaabsxr";
    char *p=strchr(base,x);
    if (*p) return cbase[p-base];
    return 0;
}

void
skipprint(x) int x; {
    static int c=0;
    if (x==0) c=0;
    else if (x=='\n'){
        if (c) {puts(""); c=0;}
    }else if (isalpha(x)){
        if (c>=60){puts(""); c=0;}
        else if (c>0 && c % 3==0) putchar(' ');
        putchar(x);
    }
}

```

```

        ++c;
    }
}

int check(t)btree*t;{
    if (t->data==0) return 0;
    if (strcmp(t->data->acc,"")!=0 && t->data->seq==0) return 0;
    if (t->left) if (!check(t->left)) return 0;
    if (t->right) if (!check(t->right)) return 0;
    return 1;
}

void
show(t)btree*t;{
    int i;

    /* show left arm */
    if (t->left) {
        show(t->left);
        free(t->left);
    }

    /* show data body */
    if (t->data->s < t->data->e)
        for (i=t->data->s-1; i<t->data->e; ++i)
            skipprint(t->data->seq[i]);
    else
        for (i=t->data->s-1; i>=t->data->e-1; --i)
            skipprint(complement(t->data->seq[i]));
    if (strcmp(t->data->acc,"")!=0) free(t->data->seq);
    free(t->data);

    /* show right arm */
    if (t->right) {
        show(t->right);
        free(t->right);
    }
}

/* to determine which genetic code to use */
int
SGC(b,mt)char*b;int mt;{
    switch (mt){
        case 0:
            if (strstr(b,"Mycoplasma")){
                return SGC3; /* In SODHO, this is SGC4. */
            }else if (strstr(b,"Dasycladaceae")){
                return SGC5;
            }
        Case 1:
            if (strstr(b,"Vertebrata")){
                return SGC1;
            }else if (strstr(b,"Yeast")){
                return SGC2;
            }else if (strstr(b,"Mold")){
                return SGC3;
            }else if (strstr(b,"Invertebrate")){
                return SGC4;
            }else if (strstr(b,"Protozoa")){
                return SGC6;
            }else if (strstr(b,"Plantae")){
                return SGC7;
            }else if (strstr(b,"Echinoderm")){

```



```

        return SGC8;
    }
    Case 2:
    if (strstr(b,"Cili")){
        return SGC5; /* In SODHO, this is SGC6 */
    }else if (strstr(b,"Euplotid")){
        return SGC9;
    }
}
return SGC0;
}

comment(s)char*s;{
    register char *p;
    if (p=strstr(yystr,s)){
        putchar('#');
        while (*p && (p[0]!='#' || p[1]!='/')) putchar(*p++);
    }
}

parse(str)char*str;{
    int n=0; /* #of CDS */
    char *p,*q;
    char buf[80];

    /* skip the first CDS_SEPERATOR. */
    parsestr=yystr=str+1;

    while (*yystr){
        if (q=strchr(yystr,CDS_SEPERATOR)) *q=0;
#ifdef comment
        if ((strchr(yystr,'<')!=0) || (strchr(yystr,'>')!=0)){
            ERRORS++;
            /**/if(debug)fprintf(stderr,"Skip %s: %s\n",ent.locus,yystr);
            if (q){
                parsestr=yystr=q+1;
                continue;
            }else return;
        }
#endif
        if (strchr(yystr,'<')==0 &&
            strchr(yystr,'>')==0 &&
            strstr(yystr,"/partial")==0 &&
            strstr(yystr,"/pseudo")==0){
            n++;

            PRINTED_CDS++;
            /* locus line */
            sprintf(buf,">%s-%d",ent.locus,n);
            printf("%-15s %s %s\n",buf,ent.div,ent.acc);

            /* print the definition and the used genetic code */
            printf("%s",ent.def);
#ifdef GENETIC_CODE
            printf("#%s\n",sgcstr[ent.sgc]);
#endif

            /* CDS region information */
            putchar('#');
            for (p=yystr; *p && (p[0]!='#' || p[1]!='/'); putchar(*p++));

            /* qualifiers */
            comment("gene=");

```

```

comment("product=");
comment("note=");

/* parse CDS line and construct the binary tree */
root=0;
yyvsparse();

/* show sequence */
skipprint(0);
if (root && check(root)) show(root);
skipprint('\n');
puts("//");
}else{
    /**/if(debug)fprintf(stderr,"Skip %s: %s\n",ent.locus,yystr);
    ERRORS++;
}
if (q) parsestr=yystr=q+1;
else return;
}
}

enum {NONE,DEFINITION,SOURCE,CDS,SEQUENCE};

run(){
    int i,mt,mode=NONE;

    while (fgets(b,BUFSIZ+1,f)){
        if (isspace(b[0])){
            if (strncmp(b+5,"CDS",3)==0){
                TOTAL_CDS++;
                /*if (debug) fprintf(stderr,"%s",b);*/
                strcat(ent.cds,CDS_SEP_STR);
                strcat(ent.cds,b+21);
                mode=CDS;
                continue;
            }
            switch(mode){
                case CDS:
                    if (isspace(b[5])){
                        /*if (debug) fprintf(stderr,"%s",b);*/
                        strcat(ent.cds,"#");
                        strcat(ent.cds,b+21);
                    }
                    Case SEQUENCE:
                    for(i=10;b[i]!='\n';i++){
                        if (isalpha(b[i])) ent.seq[ent.nseq++]=b[i];
                    }
                    ent.seq[ent.nseq]=0;
                    Case SOURCE:
                    if (ent.sgc==0) ent.sgc=SGC(b,mt);
                    Case DEFINITION:
                    strcat(ent.def,"#");
                    strcat(ent.def,b+12);
                }
            }else if (strncmp(b,"LOCUS",5)==0){
                TOTAL++;
                /* init variables */
                for (i=0;isalnum(ent.locus[i]=b[i+12]);++i);
                ent.locus[i]=0;
                for (i=0;i<Ndiv-1;++i) ent.div[i]=b[i+52]; ent.div[i]=0;
                for (i=22; !isdigit(b[i]); ++i);
                ent.size=b[i]-'0';
                while (isdigit(b[++i])) ent.size=10*ent.size+b[i]-'0';
                mode=NONE;
            }
        }
    }
}

```

```

    *ent.cds=0;
    ent.ncds=0;
    /*if (debug) fprintf(stderr,"%s",b);*/
} else if (strcmp(b,"ACCESSION",9)==0){
    for (i=0;i<Nacc-1;++i) ent.acc[i]=b[i+12]; ent.acc[i]=0;
    /*if (debug) fprintf(stderr,"%s",b);*/
} else if (strcmp(b,"DEFINITION",10)==0){
    strcpy(ent.def,"#def=");
    strcat(ent.def,b+12);
    mode=DEFINITION;
    /*if (debug) fprintf(stderr,"%s",b);*/
} else if (strcmp(b,"SOURCE",6)==0){
    ent.sgc=0;
    mt=0;
    /* check mitochondria */
    if (strstr(ent.def,"itochondrial") || strstr(ent.def," MT ") ||
        strstr(ent.def," Mt ") || strstr(ent.def," mt ")){
        mt=1;
    /* check macronuclear */
    } else if (strstr(ent.def,"acronuclear")){
        mt=2;
    }
    ent.sgc=SGC(ent.def,mt);
    if (ent.sgc==0) ent.sgc=SGC(b,mt);
    mode=SOURCE;
    /*if (debug) fprintf(stderr,"%s",b);*/
} else if (strcmp(b,"ORIGIN",6)==0){
    mode=SEQUENCE;
    /*ent.seq=(char*)malloc(sizeof(char)*ent.size+1);*/
    ent.nseq=0;
    /*if (debug) fprintf(stderr,"%s",b);*/
} else if (strcmp(b,"/",2)==0){
    /*if (debug) {
        fprintf(stderr,"Parsing CDS:");
        fprintf(stderr,"%s\n",ent.cds);
    }*/
    if (*ent.cds){
        /*if (debug){
            fprintf(stderr,"%-15s %s %s %d bp fp: %d - ",
                ent.locus,ent.div,ent.acc,ent.nseq,ftell(f));
        }*/
        parse(ent.cds);
        /*if (debug) fprintf(stderr,"Done.\n");*/
    }
} else{
    mode=NONE;
}
}

main(argc,argv){int argc; char*argv[];{
    int i;
    debug=getenv("cdsldebug");
    if (argc<2){
        fprintf(stderr,CREDIT);
        fprintf(stderr,"Usage: %s file [file-.]\\n\\n",argv[0]);
    }
    for (i=1; i<argc; ++i){
        if (f=fopen(argv[i],"r")){
            TOTAL=0;
            TOTAL_CDS=0;
            PRINTED_CDS=0;
            ERRORS=0;

```

```
run();

fprintf(stderr, "Total entries:          %8d\n", TOTAL);
fprintf(stderr, "Total CDS found:         %8d\n", TOTAL_CDS);
fprintf(stderr, "Total CDS printed:      %8d\n", PRINTED_CDS);
fprintf(stderr, "Excluded cds:           %8d\n", ERRORS);
}else
    fprintf(stderr, "cannot open %s\n", argv[i]);
}
exit(0);
}
```

aln (perl) - reformat alignment.

```
#!/usr/bin/perl -w
```

```
require 5;
$NAMELEN  = 14;
$WIDTH    = 60;

while ( $_ = $ARGV[ 0 ], /^-/ ) {
    shift;
    /^(--name=)(\d+)/ and ( $NAME = $2 ), next;
    /^-n$/ and ( $NAME = shift ), next;
    /^(--width=)(\d+)/ and ( $WIDTH = $2 ), next;
    /^-w$/ and ( $WIDTH = shift ), next;
}

sub output;
sub clear;

while ( <> ) {
    next if /^\\n$/;
    print and next if /^>/;
    output, clear and next if m|^//|;

    chop;
    $name = substr( $_, 0, $NAMELEN );
    $seq  = substr( $_, $NAMELEN + 1 );

    if ( defined $seq{ $name } )
        $seq{ $name } .= $seq;
    } else {
        $seq{ $name } = $seq;
        push( @name, $name );
    }
}

sub output {
    while ( $seq{ $name[ 0 ] } ) {
        foreach ( @name ) {
            printf "%-${NAMELEN}s ", $_;
            print substr( $seq{ $_ }, 0, $WIDTH ), "\n";
            if ( length( $seq{ $_ } ) > $WIDTH ) {
                $seq{ $_ } = substr( $seq{ $_ }, $WIDTH );
            } else {
                $seq{ $_ } = '';
            }
        }
        print "\n";
    }
}

sub clear {
    undef %seq;
    @name = ();
    return 1;
}
```


dall.c (C): create coding sequence alignments using SODHO database and coding sequence database.

```

/*
  dal.c - to make cDNA alignments using amino acid alignment and cDNA
  sequence.
  revised to accept alignment format
  Toshinori Endo 1994.3.23
*/

int debug=0;

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <malloc.h>

#define Case break;case
#define NAMELEN 14
#define LINELEN 200
#define AA_PER_LINE 60
#define SEQLEN 20000
#define MAXSEQ 500
#define CODONS_PER_LINE 15
#define PTRLEN 10
#define DIVLEN 3
#define ACCLLEN 6

void
usage(fn)char*fn;{
    fprintf(stderr,"dall.c - to make cDNA alignment");
    fprintf(stderr," using cDNA file and amino acid alignments.\n");
    fprintf(stderr,"\tToshinori Endo 1994.4.11\n");
    fprintf(stderr,
        "Usage: %s [-i index_file] [-s] cDNA_file [file-]\n\n",fn);
    exit(1);
}

enum FORMAT {DETECT,DDBJ,GENBANK,EMBL,ODEN,DNA_ALIGNMENT,
             ALIGNMENT,PACKED_ALIGN1,PACKED_ALIGN2};
char
*FORMAT_STR[]={ "DETECT", "DDBJ", "GENBANK", "EMBL", "ODEN", "DNA_ALIGNMENT",
                "ALIGNMENT", "PACKED_ALIGN1", "PACKED_ALIGN2" };
char *GAP="---";

typedef struct {
    char name[NAMELEN+1];
    int len,len2,align,checked;
    char seq[SEQLEN];
    char *cDNA;
} data_type;

typedef struct{
    char name[NAMELEN+1];
    char div[DIVLEN+1];
    char acc[ACCLLEN+1];
    char ptr[PTRLEN+1];
} index;

FILE*cDNA_file,*index_file=0;
data_type Data[MAXSEQ];
char b[LINELEN];
sizeof_indexfile=0;
int Nlines=0;

```

```

int INPUT_FORMAT=ALIGNMENT;
int OUTPUT_FORMAT=DNA_ALIGNMENT;

int
strncpy(d,s,n)char*d,*s;int n;{
    register i;
    if (n>0)
        for (i=0; i<n && !isspace(s[i]); ++i)
            d[i]=s[i];
    else
        for (i=0; !isspace(s[i]); ++i)
            d[i]=s[i];
    d[i]=0;
    return i;
}

int
alignment(f,data)FILE*f;data_type *data;{
    register i;
    char key[NAMELEN+1];
    int Nseq=0,Nseq0=0;

    /* read aligned sequences and count the number of sequences */
    while (fgets(b,LINELEN,f)){
        Nlines++;
        /* comment line or space line */
        if (isspace(*b)) continue;
        if (*b=='#' || *b=='>'){
            printf("%s",b);
            if (debug) fprintf(stderr,"* Blank\n");
            continue;
        }
        if (strncmp(b,"//",2)==0){
            return Nseq0;
        }

        /* to get the entry key */
        strncpy(key,b,NAMELEN);

        /* to find the corresponding data in the array */
        for (Nseq=0;Nseq<Nseq0;++Nseq)
            if (strcmp(key,data[Nseq].name)==0){
                if (debug) fprintf(stderr,"%3d>%s\n",Nseq,key);
                break;
            }

        /* to make a space for a new entry */
        if (Nseq>=Nseq0){
            if (debug) fprintf(stderr,"* NewKey>%3d:%s\n",Nseq,key);
            Nseq0=Nseq+1;
            strcpy(data[Nseq].name,key);
            data[Nseq].len=data[Nseq].align=0;
            data[Nseq].checked=1;
        }

        /* to read sequence */
        for (i=strlen(key); b[i]; i++){
            if (!isspace(b[i])){
                if (isalpha(b[i])){
                    b[i]=toupper(b[i]);
                    data[Nseq].len++;
                }
                data[Nseq].seq[data[Nseq].align++]=b[i];
            }
        }
    }
}

```

```

    }
    }
    return Nseq0;
}

enum {QUERY, NUM, HOMOLOGUE, SEQUENCE};

int
packed_align(type, f, data) int type; FILE *f; data_type *data; {
    int i;
    int n=0, N=0;
    int mode=QUERY;

    /* read aligned sequences and count the number of sequences */
    while (fgets(b, AA_PER_LINE+1, f)) {
        Nlines++;

        /* space line - separator of the records. */
        if (*b=='\n' && type==PACKED_ALIGN1) return N;

        /* comment line - this actually is not in the SODHO. */
        if (*b=='#' || *b=='>') {
            printf("%s", b);
            if (b[strlen(b)-1]!='\n')
                while (putchar(getc(f))!='\n');
            continue;
        }

        /* cut off trailing NEWLINE */
        if (type==PACKED_ALIGN1) getchar();

        switch (mode) {
            case QUERY: /* the query entry name */
                strncpy(data[0].name, b, NAMELEN);
                data[0].len=data[0].align=0;
                data[0].checked=1;
                mode=NUM;

            Case NUM: /* the number of homologues */
                N=atoi(b)+1;
                if (N==1) {
                    n=0;
                    mode=SEQUENCE;
                } else if (N>1) {
                    n=1;
                    mode=HOMOLOGUE;
                } else {
                    fprintf(stderr, "Error in format(%d):\n%s", Nlines, b);
                    exit(1);
                }

            Case HOMOLOGUE: /* the homologue entry name */
                strncpy(data[n].name, b, NAMELEN);
                data[n].len=data[n].align=0;
                data[n].checked=1;
                if (++n>=N) {
                    n=0;
                    mode=SEQUENCE;
                }

            Case SEQUENCE: /* the aligned sequence */
                for (i=0; i<AA_PER_LINE && !isspace(b[i]); i++) {
                    data[n].seq[data[n].align++]=b[i];
                }
        }
    }
}

```

```

        if (isupper(b[i])) data[n].len++;
    }
    ++n;
    if (n>=N && type==PACKED_ALIGN2
        && (strlen(b)<AA_PER_LINE || b[AA_PER_LINE-1]=='\n')){
        return N;
    }else if (n>=N)
        n=0;
    }
}
return N;
}

int
oden(f,seq,len)FILE*f; char *seq; int len;{
    int i,c=0;

    while (fgets(b,LINELEN,f)){
        if (*b=='#' || *b=='>'){
            printf("%s",b);
            continue;
        }
        if (b[0]=='/' && b[1]=='/') break;
        for (i=0;b[i] && c<len;i++){
            if (isalpha(b[i]))
                seq[c++]=tolower(b[i]);
        }
    }
    return c;
}

long
fsize(f)FILE*f;{
    long fp=ftell(f),fs;
    fseek(f,0,2);
    fs=ftell(f);
    fseek(f,fp,0);
    return fs;
}

/* to succeed file pointer to specified locus name */
int
cDNA_index(locus) char *locus;{
    int len=strlen(locus);
    int rec,ptr1,ptr2,d;
    index idx;

    if (!index_file){
        rewind(cDNA_file);
        while (fgets(b,LINELEN,cDNA_file)){
            if (*b=='>' && strncmp(b+1,locus,len)==0){
                printf("%s",b);
                return 1;
            }
        }
    }else{
        ptr1=0;
        ptr2=(sizeof_indexfile+1)/sizeof(index);
        while (ptr1<=ptr2){
            rec=(ptr1+ptr2)/2;
            fseek(index_file,rec*sizeof(index),0);
            fread(&idx,sizeof(index),1,index_file);
            d=strncmp(idx.name,locus,len);

```

```

        if (d==0){
            fseek(cDNA_file,atol(idx.ptr),0);
            printf(">%.s%.s %.s\n",NAMELEN,idx.name,
                DIVLEN,idx.div,
                ACCLEN,idx.acc);

            return 1;
        }else if (d<0){
            ptr1=(ptr1+ptr2+2)/2;
        }else if (d>0){
            ptr2=(ptr1+ptr2-2)/2;
        }
    }
}
return 0;
}

int
corresponding_sequences(n,data)int n;data_type*data;{
    int i,c=0;

    for (i=0; i<n; ++i){
        if (cDNA_index(data[i].name)){
            data[i].cDNA=(char*)malloc(3*data[i].len+1);
            if (!data[i].cDNA)
                fprintf(stderr,"memory overflow.\n");
            data[i].len2=oden(cDNA_file,data[i].cDNA,3*data[i].len);
            c++;
        }else{
            data[i].checked=0;
            data[i].cDNA=0;
            fprintf(stderr,"%s(%d): Can't find %s\n",
                data[0].name,n,data[i].name);
        }
    }
    return c;
}

struct {char *Codon, AA;} CodTab[] =
{
    "ttt",'F',"tct",'S',"tat",'Y',"tgt",'C',
    "ttc",'F',"tcc",'S',"tac",'Y',"tgc",'C',
    "tta",'L',"tca",'S',"taa",'*', "tga",'*',
    "ttg",'L',"tcg",'S',"tag",'*', "tgg",'W',
    "ctt",'L',"cct",'P',"cat",'H',"cgt",'R',
    "ctc",'L',"ccc",'P',"cac",'H',"cgc",'R',
    "cta",'L',"cca",'P',"caa",'Q',"cga",'R',
    "ctg",'L',"ccg",'P',"cag",'Q',"cgg",'R',
    "att",'I',"act",'T',"aat",'N',"agt",'S',
    "atc",'I',"acc",'T',"aac",'N',"agc",'S',
    "ata",'I',"aca",'T',"aaa",'K',"aga",'R',
    "atg",'M',"acg",'T',"aag",'K',"agg",'R',
    "gtt",'V',"gct",'A',"gat",'D',"ggt",'G',
    "gtc",'V',"gcc",'A',"gac",'D',"ggc",'G',
    "gta",'V',"gca",'A',"gaa",'E',"gga",'G',
    "gtg",'V',"gcg",'A',"gag",'E',"ggg",'G'
};

#define isbase(x) (x=='t' || x=='c' || x=='a' || x=='g')
#define aa(a,b,c) (CodTab[a*16+c*4+b].AA)
#define base2num(x) ((x=='t')?0:(x=='c')?1:(x=='a')?2:(x=='g')?3:4)

int
check_sequences(n,data)int n;data_type*data;{
    int i,a,c,x=n;

```



```

for (i=0;i<n;i++){
    c=0;
    if (!data[i].checked) continue;
    for (a=0; a<data[i].align; ++a){
        if (c>data[i].len2){
            data[i].checked=0;
            x--;
            fprintf(stderr,"%s: %s: Error in size\n",
                data[0].name,data[i].name);
            break;
        }
        if (isupper(data[i].seq[a])
            &&isbase(data[i].cDNA[c])
            &&isbase(data[i].cDNA[c+1])
            &&isbase(data[i].cDNA[c+2])){
            if (aa(base2num(data[i].cDNA[c]),
                base2num(data[i].cDNA[c+1]),
                base2num(data[i].cDNA[c+2])))
                == data[i].seq[a])
                c+=3;
            else{
                data[i].checked=0;
                x--;
                fprintf(stderr,"%s:[%d/%d]%s:the codon for '%c(%d)' ",
                    data[0].name,i+1,n,data[i].name,data[i].seq[a],c+1);
                fprintf(stderr,"represents '%.3s(%c)'.\n",
                    data[i].cDNA+c,
                    aa(base2num(data[i].cDNA[c]),
                        base2num(data[i].cDNA[c+1]),
                        base2num(data[i].cDNA[c+2])));
                break;
            }
        }
    }
}
return x;
}

void
print_sequence(name, seq)char name[],seq[];{
    int i,j;
    int len=strlen(seq);
    printf( ">%15s\n", name);
    for (j=0; j<len; j+=45){
        for (i=0; i+j<len && i<45; i+=3){
            printf("%.3s ", &seq[j+i]);
        }
        puts("");
    }
}

print_sequences(n,data)int n; data_type data[];{
    int i;
    for (i=0; i<n; i++){
        print_sequence(data[i].name, data[i].seq);
    }
}

#define W_ALIGNMENT
#ifdef W_ALIGNMENT
void
print_alignment(n,data)int n;data_type*data;{

```

```

int i,j,c;

for (c=0;c<data[0].align;c+=AA_PER_LINE){
    for (i=0;i<n;++i){
        if (!data[i].seq) continue;
        if (!data[i].checked) continue;
        printf("%-*s",NAMELEN-1,data[i].name);
        for (j=0; j<AA_PER_LINE && c+j<data[0].align;
            if (j % 10==0) putchar(' ');
            putchar(data[i].seq[c+j]);
        }
        puts("");
    }
    if (j>=AA_PER_LINE)
        puts("");
    else
        puts("//");
}
}
#endif

#ifdef W_PAKED_ALIGN
void
print_packed_align(type,n,data)int type,n;data_type*data;{
    int i,j,c=0;

    if (!data[0].seq || !data[i].checked) return;
    for (i=1;i<n;++i)
        if (data[i].seq && data[i].checked) c++;

    printf("%-*s\n%4d\n",NAMELEN,data[0].name,c);
    for (i=1;i<n;i++){
        if (data[i].seq && data[i].checked)
            printf("%-*s",NAMELEN,data[i].name);

        for (c=0;c<data[0].align;c+=AA_PER_LINE){
            for (i=0;i<n;++i){
                if (!data[i].seq) continue;
                for (j=0; j<AA_PER_LINE && c+j<data[i].align;
                    putchar(data[i].seq[data[i].align]);
                }
                if (type==PACKED_ALIGN1) puts("");
                else if (type==PACKED_ALIGN2 && j<AA_PER_LINE) puts("");
            }
            if (j>=AA_PER_LINE)
                puts("");
            else
                puts("//");
        }
    }
}
#endif

void
print_cdna_alignment(n,data)int n;data_type*data;{
    int i,j,c;
    int L[MAXSEQ];

    for (i=0;i<n;++i)
        L[i]=0;

    for (c=0;c<data[0].align;c+=CODONS_PER_LINE){
        for (i=0;i<n;++i){
            if (!data[i].checked) continue;

```

```

printf("%-*s",NAMELEN,data[i].name);

for (j=0; j<CODONS_PER_LINE && c+j<data[i].align;
    if (L[i]<=data[i].len2
        && isalpha(data[i].seq[c+j])){
            printf(" %.3s",data[i].cDNA+L[i]);
            L[i]+=3;
        }else{
            printf(" %.3s",GAP);
        }
    }
    puts("");
}
if (c+j<data[0].align){
    puts("");
}else{
    puts("//");
    for (i=0; i<n; ++i){
        if (data[i].cDNA){
            free(data[i].cDNA);
        }
    }
}
}

void
report(n,s,d)int n;char*s;data_type*d;{
    int i;
    fprintf(stderr,"* report> the number of the members is %d\n",n);
    for (i=0;i<n;i++){
        fprintf(stderr,"* %s%s:\t[%d/%d] %s(aln=%d, cds=%d)\n",
            d[0].name,s,i,n,d[i].name,d[i].align,d[i].len2);
    }
}

int
read_alignment(x,f,data)int x; FILE*f; data_type*data;{
    switch (x){
        case PACKED_ALIGN2:
            return packed_align(PACKED_ALIGN2,stdin,data);
        case ALIGNMENT:
            return alignment(stdin,data);
    }
}

void
run(){
    int n;
    while (n=read_alignment(INPUT_FORMAT,stdin,Data)){
        if (n==1) continue;
        /*printf("# member: %d\n",n);*/
        /*print_alignment(n,Data);*/
        print_sequences(n,Data);/*
        corresponding_sequences(n,Data);
        check_sequences(n,Data);
        print_cDNA_alignment(n,Data);
        if (debug) {
            fprintf(stderr,"* %d lines. Latest data is as follows.\n",Nlines);
            report(n,"L>",Data);
        }
    }
}

```

```

void
main(argc,argv)int argc;char*argv[];{
    int i,j,c=-1;

    if (argc<2){
        usage(argv[0]);
        exit(2);
    }

    if (getenv("daldebug")) debug=1;

    for (i=1;i<argc;i++){
        if (*argv[i]=='-'){
            switch (argv[i][1]){
                case 'i':
                    if (i+1<argc){
                        if (!(index_file=fopen(argv[++i],"r"))){
                            fprintf(stderr,"Can't open %s.\n",argv[i]);
                            exit(1);
                        }
                        sizeof_indexfile=fsize(index_file);
                        if (debug) fprintf(stderr,"* index file: %s\n",argv[i]);
                    }else
                        usage(argv[0]);
                    Case 's':
                        INPUT_FORMAT=PACKED_ALIGN2;
                }
                continue;
            }
            c++;
            if (c==0){
                if (!(cdna_file=fopen(argv[i],"r"))){
                    fprintf(stderr,"Can't open cDNA file: %s.\n",argv[i]);
                    exit(1);
                }
                if (debug) fprintf(stderr,"* cDNA file: %s\n",argv[i]);
                continue;
            }
            if (freopen(argv[i],"r",stdin)){
                if (debug){
                    fprintf(stderr,"* input file: %s\n",argv[i]);
                    fprintf(stderr,"* read mode: %s\n",FORMAT_STR[INPUT_FORMAT]);
                    fprintf(stderr,"* write
mode: %s\n",FORMAT_STR[OUTPUT_FORMAT]);
                }
                run();
            }else{
                fprintf(stderr,"Can't open %s.\n",argv[i]);
            }
        }
    }
    if (!c){
        if (debug) fprintf(stderr,"* (stdin)\n");
        run();
    }
    exit(0);
}

```

distng.c (C): calculating the numbers of synonymous and nonsynonymous substitutions between pairs of sequence properly aligned.

```

/*
  distng.c - to calculate the numbers of synonymous and nonsynonymous
             substitution by Nei and Gojobori's method.
  Toshinori Endo  1994.3.8 4.16 5.11
*/

#include <stdio.h>
#include <ctype.h>
#include <math.h>

#define INFDOUBLE (0x7FF0000000000000U)
#define Case break;case
#define Default break;default

usage(str)char*str;{
  fprintf(stderr,"%s\n%s\n%s\n\n",
    "distng.c - to estimate the numbers of synonymous and nonsynonymous",
    "\tsubstitutions by Nei and Gojobori's method (1986)",
    "\tToshinori Endo  1994.5.11");
  fprintf(stderr,"Usage: %s [-sh] [file-]\n",str);
  fprintf(stderr,"\toptions s: suppress header lines;  h: help\n\n");
}

#define NAMELEN 20
#define LINELEN BUFSIZ
#define SEQLEN 20000
#define MAXSEQ 500

typedef struct {
  char *cod,aa;
  double psyn;
}CodonTable;

/* codon table for standard genetic code */
static CodonTable CodTab[64]={
  "ttt",'F',1.0/3.0, "tct",'S',1.0, "tat",'Y',1.0/3.0, "tgt",'C',1.0/3.0,
  "ttc",'F',1.0/3.0, "tcc",'S',1.0, "tac",'Y',1.0/3.0, "tgc",'C',1.0/3.0,
  "tta",'L',2.0/3.0, "tca",'S',1.0, "taa",'*',0.0, "tga",'*',0.0,
  "ttg",'L',2.0/3.0, "tcg",'S',1.0, "tag",'*',0.0, "tgg",'W',0.0,
  "ctt",'L',1.0, "cct",'P',1.0, "cat",'H',1.0/3.0, "cgt",'R',1.0,
  "ctc",'L',1.0, "ccc",'P',1.0, "cac",'H',1.0/3.0, "cgc",'R',1.0,
  "cta",'L',4.0/3.0, "cca",'P',1.0, "caa",'Q',1.0/3.0, "cga",'R',4.0/3.0,
  "ctg",'L',4.0/3.0, "ccg",'P',1.0, "cag",'Q',1.0/3.0, "cgg",'R',4.0/3.0,
  "att",'I',2.0/3.0, "act",'T',1.0, "aat",'N',1.0/3.0, "agt",'S',1.0/3.0,
  "atc",'I',2.0/3.0, "acc",'T',1.0, "aac",'N',1.0/3.0, "agc",'S',1.0/3.0,
  "ata",'I',2.0/3.0, "aca",'T',1.0, "aaa",'K',1.0/3.0, "aga",'R',2.0/3.0,
  "atg",'M',0.0, "acg",'T',1.0, "aag",'K',1.0/3.0, "agg",'R',2.0/3.0,
  "gtt",'V',1.0, "gct",'A',1.0, "gat",'D',1.0/3.0, "ggt",'G',1.0,
  "gtc",'V',1.0, "gcc",'A',1.0, "gac",'D',1.0/3.0, "ggc",'G',1.0,
  "gta",'V',1.0, "gca",'A',1.0, "gaa",'E',1.0/3.0, "gga",'G',1.0,
  "gtg",'V',1.0, "gcg",'A',1.0, "gag",'E',1.0/3.0, "ggg",'G',1.0
};

typedef struct {
  char *cod1,*cod2;
  double syn, nsy;
}CodonTable2;

static CodonTable2 CodTab2[4096]={
#include "syn.tab"
};

```



```

typedef struct {
    char name[NAMELEN];
    int len, align;
    char seq[SEQLEN];
} data_type;

data_type data[MAXSEQ];
char b[LINELLEN];
static char *bases="acgt";
int noheader=0;

#define conv(x) ((x=='t')?0:(x=='c')?1:(x=='a')?2:(x=='g')?3:-1)

char aa(s)char s[3];{
    return CodTab[conv(s[0])*16+conv(s[1])+conv(s[2])*4].aa;
}

#define iscodon(s) (strchr(bases,*s)!=0 && strchr(bases,*s+1)!=0 && \
    strchr(bases,*s+2)!=0)
#define jc(p) ((1.0-p*4.0/3.0 >0)?(-log(1.0-p*4.0/3.0)*3.0/4.0):1e1000)

/* Attention!! Following macros should be used after iscodon() checking */
#define pos_syn(s) \
    ((double)CodTab[conv(*s)*16+conv(*s+2)*4+conv(*s+1)]).psyn)
#define obs_syn(s,t) \
    CodTab2[conv(*s+1)*1024+conv(*s)*256+conv(*s+2)*64\
    +conv(*t+1)*16+conv(*t)*4+conv(*t+2)].syn
#define obs_nsy(s,t) \
    CodTab2[conv(*s+1)*1024+conv(*s)*256+conv(*s+2)*64\
    +conv(*t+1)*16+conv(*t)*4+conv(*t+2)].nsy

double
value(str)char*str;{
    double d=0.0, dl=0.0, up=0.0;
    int sign=1;

    if (*str=='-') sign=-1,*str++;
    while (*str){
        if (isdigit(*str) && up==0.0) d = d*10 + (*str-'0');
        else if (isdigit(*str) && up>0) d += up* (*str-'0');
        else if (*str=='.' && up==0.0) up=0.1;
        else if (*str=='.' && up>0) up /= 10.0;
        else if (*str==',');
        else if (*str=='/') dl=d,d=0.0,up=0.0;
        else break;
        str++;
    }
    d *= sign;
    return (dl==0.0)? d: (d!=0.0)? dl/d: INFDDOUBLE;
}

int
Table1(b,codon)char*b;int codon;{
    int i=0;
    while (b[i]){
        /* codon */
        while (b[i] && !isalpha(b[i])) i++;
        if (!b[i]) break;
        if ((CodTab[codon].cod[0]=tolower(b[i]))=='u')
            CodTab[codon].cod[0]='t';
        i++;
    }
}

```

```

    if ((CodTab[codon].cod[1]=tolower(b[i]))=='u')
        CodTab[codon].cod[1]='t';
    i++;
    if ((CodTab[codon].cod[2]=tolower(b[i]))=='u')
        CodTab[codon].cod[2]='t';
    i++;

    /* one letter amino acid symbol */
    while (!isalpha(b[i]) && b[i]!='*') i++;
    CodTab[codon].aa=toupper(b[i]); i++;

    /* candidate synonymous substitution numbers */
    while (!isalpha(b[i])) i++;
    CodTab[codon].psyn=value(b+i);
    while (isdigit(b[i])||strchr(",.-/",b[i])) i++;
    codon++;
}
return codon;
}

int
Table2(b,codon)char*b;int codon;{
    int i=0;
    while (b[i]){
        /* codon1 */
        while (b[i] && !isalpha(b[i])) i++;
        if (!b[i]) break;

        if ((CodTab2[codon].cod1[0]=tolower(b[i]))=='u')
            CodTab2[codon].cod1[0]='t';
        i++;
        if ((CodTab2[codon].cod1[1]=tolower(b[i]))=='u')
            CodTab2[codon].cod1[1]='t';
        i++;
        if ((CodTab2[codon].cod1[2]=tolower(b[i]))=='u')
            CodTab2[codon].cod1[2]='t';
        i++;
        while (b[i] && !isalpha(b[i])) i++;
        if (b[i]=='(') while (b[++i]!='('));

        /* codon2 */
        while (b[i] && !isalpha(b[i])) i++;

        if ((CodTab2[codon].cod2[0]=tolower(b[i]))=='u')
            CodTab2[codon].cod2[0]='t';
        i++;
        if ((CodTab2[codon].cod2[1]=tolower(b[i]))=='u')
            CodTab2[codon].cod2[1]='t';
        i++;
        if ((CodTab2[codon].cod2[2]=tolower(b[i]))=='u')
            CodTab2[codon].cod2[2]='t';
        i++;
        while (b[i] && !isalpha(b[i])) i++;
        if (b[i]=='(') while (b[++i]!='('));

        /* codon to codon synonymous substitution numbers */
        while (!isalpha(b[i])) i++;
        CodTab2[codon].syn=value(b+i);
        while (isdigit(b[i])||strchr(",.-/\\t",b[i])) i++;

        /* codon to codon nonsynonymous substitution numbers */
        while (!isalpha(b[i])) i++;
        CodTab2[codon].nsy=value(b+i);
    }
}

```

```

        while (isdigit(b[i]) || strchr(",.-/\t",b[i])) i++;
        codon++;
    }
    return codon;
}

int
ReadGeneticCode(GeneticCode,Tab) char*GeneticCode;int Tab;{
    FILE *f=fopen(GeneticCode,"r");
    char b[LINELLEN];
    int flag=0,Table=Tab,tab1=0,tab2=0;
    int i,codon;

    if (!f){
        fprintf(stderr,"Can't open %s.\n",GeneticCode);
        exit(2);
    }
    while (fgets(b,LINELLEN,f)){
        if (*b=='>'){flag=1;continue;}
        if (!flag) continue;
        if (*b=='\n') continue;
        if (*b=='#'){
            if (b[1]=='1' || b[1]=='2'){
                Table==b[1]-'0';
                codon=0;
            }
            else continue;
        }
        switch (Table){
            case 1:
                codon=Table1(b,codon);
                if (codon>=64) tab1=1;
                break;
            case 2:
                codon=Table2(b,codon);
                tab2=1;
                if (codon>=4096) tab2=1;
        }
        switch (Tab){
            case 0: if (tab1 || tab2) return 0;
            case 1: if (tab1) return 0;
            case 2: if (tab2) return 0;
            case 3: if (tab1 && tab2) return 0;
        }
    }
    return EOF;
}

title_header(Nseq0) int Nseq0;{
    int i,j,k,a,c,g,t;
    double ps;

    puts("# NG method estimation of the numbers of synonymous and
nonsynonymous change");
    puts("#");
    printf("# the number of sequences : %d\n",Nseq0);
    printf("# sequence          nt sites syn.sites nsy.sites      a      c
g      t      %%GC\n");
    /* count the synonymous sites */
    for (i=0; i<Nseq0; i++){
        /*          printf("# ");*/
        ps=0.0; a=c=g=t=0;

```

```

    for (k=0; k<data[i].align; k+=3){
        if (iscodon(data[i].seq+k)){
            ps+=pos_syn(data[i].seq+k);
            putchar(aa(data[i].seq+k)); */
        }
        for (j=0; j<3; j++){
            switch (data[i].seq[k+j]){
                case 'a': a++;
                case 'c': c++;
                case 'g': g++;
                case 't': t++;
            }
        }
    }
    puts(""); */

    printf("# %-15.15s %5d %5d %8.2f %8.2f, %4d %4d %4d %4d",
        data[i].name, data[i].len, data[i].align, ps, data[i].len-ps,
        a, c, g, t);
    printf(" %.2f\n", 100.0*(double)(g+c)/(double)(a+c+g+t));
}

puts("#");
puts("# Psyn, Pnsy: possible synonymous and nonsynonymous sites");
puts("# Osyn, Onsy: observed synonymous and nonsynonymous
substitutions");
puts("# Esyn, Ensy: estimated synonymous and nonsynonymous
substitutions");
printf("# No. of sequence = %d\n", Nseq0);
puts("#");
puts("# sequence1 sequence2 Psyn Osyn Esyn Pnsy Onsy
Ensy CI");
}

void
estimation(Nseq0) int Nseq0;{
    /* calculate real synonymous and nonsynonymous changes and print*/
    double ps, pn; /* possible synonymous and nonsynonymous site */
    double os, on; /* observed synonymous and nonsynonymous change */
    double es, en; /* estimated synonymous and nonsynonymous change */
    int i, j, k, base;

    /* variables for statistical analysis */
    int n=0, nv1=0, nv2=0; /* the number of data, valid data */
    double ts1=0.0, ts2=0.0, tn1=0.0, tn2=0.0; /* total */
    double ss1=0.0, ss2=0.0, sn1=0.0, sn2=0.0; /* square */
    double mc1=0.0, mc2=0.0; /* multiples */

    for (i=0; i<Nseq0; i++){
        for (j=i+1; j<Nseq0; j++){
            os=0.0; on=0.0; ps=0.0; pn=0.0; base=0;
            for (k=0; k<data[i].align; k+=3){
                if (iscodon(data[i].seq+k)&&iscodon(data[j].seq+k)){
                    base+=3;
                    ps+=(pos_syn(data[i].seq+k)+pos_syn(data[j].seq+k))/2.0;
                    pn+=3-(pos_syn(data[i].seq+k)+pos_syn(data[j].seq+k))/2.0;
                    os+=obs_syn(data[i].seq+k, data[j].seq+k);
                    on+=obs_nsy(data[i].seq+k, data[j].seq+k);
                }
            }
        }
        printf(
            "%-14.14s%-14.14s%7.2f %7.2f %7.4f %7.2f %7.2f %7.4f %7.4f\n",
            data[i].name, data[j].name,

```

```

        ps,os,es=jc(os/ps), pn,on,en=jc(on/pn), en/es);
n++;
if (es<=1.0 &&
    nv1++;
    ts1+=es;    tn1+=en;
    ssl+=es*es; sn1+=en*en;
    mcl+=es*en;
}
if (es<=2.0 && en<=2.0){
    nv2++;
    ts2+=es;    tn2+=en;
    ss2+=es*es; sn2+=en*en;
    mc2+=es*en;
}
}
}
if (noheader) return;
printf("#profile of %s group\n",data[0].name);
printf("# N=%d\n",n);
printf("# N(syn,nsy<1.0)=%d\n",nv1);
printf("# Ave.s(1.0)=%6.4f, Ave.n(1.0)=%6.4f\n",ts1/n,tn1/n);
printf("# Vs(1.0)=%6.4f, Vn(1.0)=%6.4f, Cov.(1.0)=%6.4f\n",
    ssl/n-ts1*ts1/(n*n),sn1/n-tn1*tn1/(n*n),mcl/n-ts1*tn1/(n*n)
);
puts("#");
printf("# N(syn,nsy<2.0)=%d\n",nv2);
printf("# Ave.s(2.0)=%6.4f, Ave.n(2.0)=%6.4f\n",ts2/n,tn2/n);
printf("# Vs(2.0)=%6.4f, Vn(2.0)=%6.4f, Cov.(2.0)=%6.4f\n",
    ss2/n-ts2*ts2/(n*n),sn2/n-tn2*tn2/(n*n),mc2/n-ts2*tn2/(n*n)
);
}

int
strspcpy(d,s,n)char*d,*s;int n;{
    register i;
    if (n>0)
        for (i=0; i<n && !isspace(s[i]); ++i)
            d[i]=s[i];
    else
        for (i=0; !isspace(s[i]); ++i)
            d[i]=s[i];
    d[i]=0;
    return i;
}

int
ReadAlignment(){
    register i;
    char key[NAMELEN+1];
    int Nseq=0,Nseq0=0;

    /* read aligned sequences and count the number of sequeces */
    while (fgets(b,LINELEN,stdin)){
        /* comment line or space line */
        if (isspace(*b)) continue;
        if (*b=='#' || *b=='>'){
            if (noheader==0) printf("%s",b);
            continue;
        }
        if (strncmp(b,"//",2)==0)
            return Nseq0;

        /* the entry key */

```



```

    strspcpy(key,b,NAMELEN);

    /* to find the corresponding data in the array */
    for (Nseq=0;Nseq<Nseq0;++Nseq)
        if (strcmp(key,data[Nseq].name)==0)
            break;

    /* to make a space for a new entry */
    if (Nseq>=Nseq0){
        Nseq0=Nseq+1;
        strcpy(data[Nseq].name,key);
        data[Nseq].len=data[Nseq].align=0;
    }

    /* to read sequence */
    for (i=strlen(key)+1; b[i]; i++){
        if (!isspace(b[i])){
            b[i]=tolower(b[i]);
            switch (b[i]){
                case 'a': case 'c': case 'g': case 't':
                    data[Nseq].len++;
                default:
                    data[Nseq].seq[data[Nseq].align++]=b[i];
            }
        }
        Nseq++;
    }
    return Nseq0;
}

run(){
    int output=0,n;

    while (n=ReadAlignment()){
        if (!noheader) title_header(n);
        estimation(n);
        if (!noheader) puts("//");
        output++;
    }
}

main(argc,argv){int argc;char*argv[];{
    int i,c=0;
    for (i=1;i<argc;++i){
        if (*argv[i]=='-'){
            switch (argv[i][1]){
                case 's':
                    noheader=1;
                Case 'c':
                    ReadGeneticCode(argv[++i],0);
                Case 'C':
                    ReadGeneticCode(argv[++i],3);
                Default:
                    usage(argv[0]);
                    exit(1);
            }
            continue;
        }
        c++;
        if (freopen(argv[i],"r",stdin))
            run();
        else{

```

```
        fprintf(stderr, "Can't open %s\n", argv[i]);
        usage(argv[0]);
    }
}
if (!c)
    run();
exit(0);
}
```

gettrans (Perl): get translation feature from GenBank formatted database.

```
#!/usr/bin/perl -w
# gettrans - get "translation" string in GenBank data.

$MINLEN = $MINLEN0 = 1;
$MAXLEN = 99999;
$KEYWORD = '';

while ( ( $_ = $ARGV[0] ), /^-/ ) {
    shift;
    ($KEYWORD = $_), next if /^-k/;
    ($MINLEN = $_), next if /^-m(\d+)/;
    ($MINLEN = shift), next if /^-m(\D|$)/;
    ($MAXLEN = $_), next if /^-M(\d+)/;
    ($MAXLEN = shift), next if /^-M(\D|$)/;

    print STDERR
        "Usage: $0 [-mmin_length] [-Mmax_length] [-keyword] [file -]\n";
    exit 1;
}

$MINLEN = $MINLEN0 if $MINLEN <= 0;

sub PrintSeq;

main_loop: while (<>){
    /^LOCUS +(\S+)/ && do {
        $LOCUS = $_;
        $NUMBER = 0;
    };
    / {5}CDS/ && do {
        ( $NOTE, $PRODUCT, $GENE ) = ( '' ) x 3;
    };
    m| +/note="(.+)"| and $NOTE = $_;
    m| +/product="(.+)"| and $PRODUCT = $_;
    m| +/gene="(.+)"| and $GENE = $_;
    m| +/translation="(.+)"| and do {
        $pep = $_;
        PrintSeq $pep, $NOTE, $PRODUCT, $GENE;
        next main_loop;
    };
    m| +/translation="| && do {
        chop( $pep = $_ );
        $trans = 'y';
        next main_loop;
    };
    /^ {21}/ && $trans and do {
        chop( $pep .= $_ );
        next main_loop unless $_ =~ /\n/;

        $pep =~ s/\n//;
        PrintSeq $pep, $NOTE, $PRODUCT, $GENE;
        $trans = '';
    };
    /^(BASE COUNT| \S)/ && $trans and do {
        PrintSeq $pep, $NOTE, $PRODUCT, $GENE;
        $trans = '';
    };
}

sub PrintSeq{
    my( $Pep, @Comments ) = @_ ;
```


mkidx.c (C) - makes index file for cDNA database.

```

/*
   mkidx.c - to make index file for cDNA file.
   Toshinori Endo 1994.3.1
*/

#include <stdio.h>

#define NAMELEN 14
#define LINELEN 100

typedef struct{
    char name[NAMELEN+1];
    long ptr;
} index;
char b[LINELEN];

int
strspcpy(d,s,n)char*d,*s;int n;{
    register i;
    if (n>0)
        for (i=0; i<n && !isspace(s[i]); ++i)
            d[i]=s[i];
    else
        for (i=0; !isspace(s[i]); ++i)
            d[i]=s[i];
    d[i]=0;
    return i;
}

void
run(){
    index idx;

    while (fgets(b,LINELEN,stdin)){
        if (*b=='>'){
            strcpy(idx.name,b+1,NAMELEN);
            idx.ptr=ftell(stdin);
            fwrite(idx,sizeof(index),1,stdout);
        }
    }
}

main(argc,argv)int argc;char*argv[];{
    if (argc>1){
        fprintf(stderr,"mkidx.c - to make index of cDNA file for dal.c.\n");
        fprintf(stderr,"\tToshinori Endo 1994.3.1\n");
        fprintf(stderr,"Usage %s < infile > outfile\n\n");
        exit (1);
    }
    run();
    exit (0);
}

```


pos (Perl): find the within-gene regions where positive selection may operate.

```
#!/usr/bin/perl -w
# possig - search for the gene regions subjected to positive selection.
```

```
sub Init;
sub GetOpt;
sub Main;
sub Help;
sub Significance;
sub ChkLen;
sub CheckPositiveSelection;
sub PrintSegment;
sub PrintSeq;
sub SigHandler;
sub PrintComments;
```

```
Init;
GetOpt;
Main;
```

```
sub Init {
    $window = 60;          # window size.
    $shift = 30;           # block shift size.

    $width = 60;           # print width.
    $block = 3;            # print block.
    $namelen = 14;         # entry name length.
    $tmpfile = "$0.tmp";   # temporary file name.
    $dist = '/home/may/local/ODEN/bin/dists';
    $SigName = 'Significance';
    $MarkName = 'dn > 2ds';
    $MAX_dN = 1;
    $MIN_dS = 0;

    $SIG{'INT'} = 'SigHandler';
    $SIG{'QUIT'} = 'SigHandler';
    # $SIG{__WARN__} = 'SigHandler';
    # $SIG{__DIE__} = 'SigHandler';
}

sub SigHandler {
    my ( $sig ) = @_;
    print STDERR "Break by SIG$sig\n" if $sig =~ /INT|QUIT/;
    print STDERR "Abnormal exit : $!\n" unless $sig =~ /INT|QUIT/;
    unlink $tmpfile;
    exit ( -1 );
}

sub GetOpt {
    while ( ( $_ = $ARGV[0] ), /^-/ ) {
        shift;
        /^-w/ && ( $window = $_ );
        /^-s/ && ( $shift = $_ );
        /^-l/ && ( $width = $_ );
        /^-b/ && ( $block = $_ );
        # /^-oden/ && ( $namelen = 10 );
        /^-t/ && ( $tmpfile = "$0/$tmpfile" );
        /^-d/ && ( $dist = $_ );
        /^-m/ && ( $MIN_dS = $_ );
        /^-M/ && ( $MAX_dN = $_ );
        /^-h/ && &help;
        # /^-a/ && ( $strict = 'yes' );
    }
}
```

```

}
Help unless $window > 0
    and $shift > 0
    and $width > 0
    and $tmpfile ne ''
    and $dist ne ''
    and $MAX_dN > 0;
}

sub Help {
    select STDERR;
    print <<"ENDofHELP";
Usage: $0 [-w#] [-s#] [-l#] [-b#] [-oden] [-tfile] [-dprogram] [-m#] [-M#] [-h]
Options: (default)
    w: window size      ($window)    d: distance calculation ($dist)
    s: shift length     ($shift)     m: minimam dS value      ($MIN_dS)
    l: print width      ($width)     M: MAXIMAM dN value      ($MAX_dN)
    b: block size       ($block)     t: tmpfile direcotry    ($tmpfile)
    h: print this help
}

ENDofHELP
exit 2;
}

sub Main {
    while ( <> ) {
        /\s/ and next;
        /^[>#]/ and push( @comments, $_ ), next;

        if ( m,^//|^#term, ) {
            @name > 1 and do {
                CheckPositiveSelection;
                $mark = '*' if $seq{ $SigName } =~ /\*/;
                $mark = '+' if $seq{ $SigName } =~ /\+/;
#                write STDERR;
                if ( $seq{ $SigName } =~ /[+*]/ or
                    $seq{ $MarkName } =~ /2/ ) {
                    select STDOUT;
                    PrintSeq( $SigName, $MarkName, @name );
                }
            };
            # clear variables.
            foreach ( keys %seq ) { undef $seq{ $_ }; }
            @name = ();
            @comments = ();
        } else {
            ( $name, $seq ) = split ' ', $_, 2;
            push( @name, $name ) unless defined $seq{ $name };
            $seq =~ s/\s//g;
            $seq{ $name } .= $seq;
        }
    }

    @name > 1 and CheckPositiveSelection;
    unlink $tmpfile;
}

sub CheckPositiveSelection {
    my ( $d );
    my ( $MarkSig );
    my ( $MarkDn2Ds );

```

```

my ( $len ) = ChkLen; # check if sequence lengths are all the same.

return if $len == 0;

@Sig = ( ' ' ) x $len;
@Mark = ( ' ' ) x $len;
for ( $site = 0; $site < $len; $site += $shift ) {
    # write the segment of specified site and window-size into tmpfile.
    open( TMP, "| egrep -v '----.*----' > $tmpfile" )
        or die "Can't open tmpfile. :$!";
    select TMP;
    PrintSegment( $site, $window, 10, @name );
    print "//\n";
    close TMP;

    ( $MarkDn2Ds, $MarkSig, $Nmark, @Pairs ) = Significance;
    $PairD{ $site } = join( '/', @Pairs[ 0 - $Nmark - 1 ] );
    $PairS{ $site } = join( '/', @Pairs[ $Nmark - $#Pairs ] );

    for $i ( $site - $site + $window - 1 ) {
        last if $i >= $len;
        if ( $MarkSig ne ' ' and
            $Sig[ $i ] ne '*' and
            $Sig[ $i ] ne $MarkSig ){
            $Sig[ $i ] = $MarkSig;
        }
        if ( $MarkDn2Ds ne ' ' and
            $Mark[ $i ] ne '2' and
            $Mark[ $i ] ne $MarkDn2Ds ) {
            $Mark[ $i ] = $MarkDn2Ds
        }
    }
}
$seq{ $SigName } = join( '', @Sig );
$seq{ $MarkName } = join( '', @Mark );
}

sub ChkLen {
    my( $len ) = length $seq{ $name[0] };

    foreach $name ( @name[ 1 - $#name ] ) {
        if ( length ( $seq{ $name } ) != $len ) {
            select STDERR;
            print "Size mismatch. ( $len )\n";
            foreach $name ( @name ) {
                printf " %-${namelen}s : %5d sites ", $name, length( $seq{ $name } );
                print '*' if $len != length $seq{ $name };
                print "\n";
            }
            select STDERR;
            PrintSeq( @name );
            select STDOUT;
            return 0;
        }
    }
    $len;
}

sub Significance {
    my( @Names ) = ();
    my( $Significance ) = ' ';
    my( $Dn2Ds ) = ' ';
    my( $sent1, $sent2 );

```

```

my( $ds, $dn, $vs, $vn, $ss, $sn );

open DIST, "$dist $tmpfile |"
    or die "Can't exec $dist: $!";
while ( <DIST> ) {
    /^#{20}/ and do {
        @_ = split;
        ( $sent1, $sent2 ) = @_[ 1, 3 ];
    };
    /^Synonymous/ and chop( $Reading = $ _ );
    /^Nonsynonymous/ and chop( $Reading = $ _ );
#    /^Gamma distance/ and chop ( $Reading = $ _ );

    /^\td = / and do {
        @_ = split ' ', $';
        if ( $Reading eq 'Synonymous' ) {
            ( $ds, $ss ) = @_[ 0, 2 ]
        } elsif ( $Reading eq 'Nonsynonymous' ) {
            ( $dn, $sn ) = @_[ 0, 2 ];
            next if $ds > 1.0 || $dn > 1.0;

            ( $vs, $vn ) = ( $ss ** 2, $sn ** 2 );

            $test = 0;
            $test = ( $dn - $ds ) / sqrt( $vs + $vn ) unless $vs + $vn == 0;
            if ( $test > 2.33 ) { # 1% level
                push @Names, "$sent1 x $sent2";
                $Significance = '*';
            } elsif ( $test > 1.65 && $Significance eq ' ' ) { # 5% level
                push @Names, "$sent1 x $sent2";
                $Significance = '+';
            }
            if ( $dn > $ds * 2 ) {
                push @Names2, "$sent1 x $sent2";
                $Dn2Ds = '2';
            } elsif ( $dn > $ds ) {
                push @Names2, "$sent1 x $sent2";
                $Dn2Ds = '1';
            }
        }
    };
}
close DIST;
( $Dn2Ds, $Significance, scalar @Names2, @Names2, @Names );
}

sub PrintComments {
    my( $site, $width, $block ) = @_;

    foreach $name ( $MarkName, $SigName ) {
        if ( $block > 0 ) {
            ( $seq = substr( $seq{ $name }, $site, $size ) ) =~ s/(.{$block})/$1/g;
        } else {
            ( $seq = substr( $seq{ $name }, $site, $size ) );
        }

        printf "%-${namelen}s $seq\n", $name{ $name };
        if ( $name eq $MarkName ) {
            for $i ( 0 - $width ) {
                if ( $PairD{ $site + $i } ) {
                    foreach $rem ( split( '/', $PairD{ $site + $i } ) ) {
                        print ' ' x ( $namelen + $i + 2 ), $rem;
                    }
                }
            }
        }
    }
}

```

```

    }
  }
} elseif ( $name eq $SigName ) {
  for $i ( 0 - $width ) {
    if ( $PairS{ $site + $i } ) {
      foreach $rem ( split( '/', $PairS{ $site + $i } ) ) {
        print ' ' x ( $namelen + $i + 2 ), $rem, "\n";
      }
    }
  }
}
}

sub PrintSeq {
  my( @name ) = @_;

  print @comments;

  $len = length $seq{ $name[0] };
  for ( $i = 0; $i < $len; $i+= $width ) {
    # PrintComments( $i, $width, $block );
    PrintSegment( $i, $width, $block, @name );
    print "\n";
  }
  print "//\n";
}

sub PrintSegment {
  my ( $site, $size, $block, @name ) = @_;
  my ( $seq );
  my ( $name1 );

  foreach $name ( @name ) {
    if ( $block > 0 ) {
      ( $seq = substr( $seq{ $name }, $site, $size ) ) =~ s/(.{$block})/$1/g;
    } else {
      ( $seq = substr( $seq{ $name }, $site, $size ) );
    }
    $name1 = $name;
    length $name1 > $namelen and
      substr( $name1, $namelen - 1 ) = substr( $name, -1 );
    printf "%-${namelen}s $seq\n", $name1;
  }
}

```

posblk (Perl): find the within-gene regions on which positive selection may operate.

```
#!/usr/bin/perl
# posblk - search for the target regions of positive selection.

$window = 60;          # window size.
$shift = 30;           # block shift size.

$width = 60;           # print width.
$block = 3;            # print block.
$namelen = 14;         # entry name length.
$tmpfile = "$0.tmp";   # temporary file name.
$distance = 'distng -s'; # command for distance matrix calculation.

#$debug = 'y';
#$debug2 = 'y';

dbmopen( DIC, 'selection', 0600);

main:
while ( <> ) {
    ( m!^//! || ( ( @name > 0 ) && /^#term/ ) ) && do {
        ( @name > 1 ) && &CheckPositiveSelection;
        &clear;
        next main;
    };
    /^[>#]/ && do { push(@comments,$_); next main; };

    # read sequence name.
    ( $name = substr( $_, 0, $namelen ) ) =~ s/^\s+|\s+$//g;
    next unless $name; # skip space line.

    # read sequence.
    chop;
    ( $seq = substr( $_, $namelen ) ) =~ s/\s//g;
    push( @name, $name ), $name{ $name } = '1' unless $name{ $name };
    $seq{ $name } .= $seq;

    print "$name $seq\n" if $debug;
}

unlink $tmpfile;

sub clear {
    foreach $name ( @name ) { delete $name{ $name }; delete $seq{ $name }; }
    @name = ();
    @comments = ();
}

sub CheckPositiveSelection {
    local($d);
    local($len) = &chklen; # check if sequence lengths are all the same.
    local($site);

    $debug2 && print STDERR "- $len @name\n";
    return unless $len;
    for ( $site = 0; $site < $len; $site += $shift ) {

        # write the segment of specified site and window-size into tmpfile.
        open( TMP, ">$tmpfile" ) || die "Can't open tmpfile. :$!";
        select TMP;
        &print segment( $site, $window, 10, @name );
        print "\n";
        close TMP;
    }
}
```



```

print "site $site, shift $shift, len $len\n";
#system "cat $tmpfile\n";

select STDOUT;
# exec $distance && read the results.
open( TMP, "$distance $tmpfile |" ) || die "Can't run $distance:$!";
while ( <TMP> ){
    split;
    ( $name1, $name2, $dS, $dN ) = @_[ 0, 1, 4, 7 ];
    $debug2 && print STDERR "    $name1 x $name2: $dS, $dN\n";
    if ( ( $dS > 0 ) && ( $dS * 2 < $dN ) && ( $dN < 1 ) ) {
        unless ( $pos{ $name1, $name2 } ) {
            print ">@ $name1 x $name2 : dS=$dS < dN=$dN at site $site.\n";
            $pname{ $name1 }++;
            $pname{ $name2 }++;
            $debug2 && print STDERR "x $name1 x $name2\n";
        }
        $pos{ $name1, $name2 }++;
    }
}
close TMP;
@pname = keys pname;
&print_seq(@pname) if @pname;
foreach ( keys pos ) { delete $pos{$_}; }
foreach ( keys pname ) { delete $pname{$_}; }
}

sub chklen {
    $len = length $seq{ $name[0] };
    foreach $name ( @name[ 1 - $#name ] ) {

        if ( length ( $seq{ $name } ) != $len ) {
            select STDERR;
            print "Size mismatch. ( $len )\n";
            foreach $name ( @name ) {
                printf " %-${namelen}s : %5d sites ", $name, length( $seq{ $name } );
                print '*' if $len != length $seq{ $name };
                print "\n";
            }
            &report;
            return 0;
        }
    }
    $len;
}

sub print_seq {
    local( @name ) = @_;
    print STDERR "-- print_seq\n" if $debug;

    print @comments;

    $len = length $seq{ $name[0] };
    for ( $i = 0; $i < $len; $i += $width ) {
        &print_segment( $i, $width, $block, @name );
        print "\n";
    }
    print "//\n";
}

sub print_segment {

```

```

local( $site, $size, $block, @name ) = @_;
local( $tmp );

foreach $name ( @name ) {
    ( $tmp = substr( $seq{ $name }, $site, $size ) ), next unless $block;
    ( $tmp = substr( $seq{ $name }, $site, $size ) =~ s/(.{$block})/ \1/g;
    printf "%-${namelen}s $tmp\n", $name;
}

sub write2dic {
    while ( ( $key, $val ) = each pos ) {
        $DIC{ $key } = $val;
    }
}

sub report {
    select STDERR;
    &print_seq;
    # @length = ();
    # foreach ( sort keys seq ) { push ( @length, length $seq{$_} ); }
    # print 'n> ', join ( ' ', sort @name ), "\n";
    # print 'N> ', join ( ' ', sort keys name ), "\n";
    # print 's> ', join ( ' ', sort keys seq ), "\n";
    # print 'l> ', join ( ' ', @length ), "\n//\n";
    # print join ( ' ', @comments ), "//\n";
    select;
}

```



```

my ($d);
my ($len) = &chklen; # check if sequence lengths are all the same.
my ($mark);

return if $len == 0;

@Sig = ( ' ' ) x $len;
for ( $site = 0; $site < $len; $site += $shift ) {
    # write the segment of specified site and window-size into tmpfile.
    open( TMP, ">$tmpfile" ) || die "Can't open tmpfile. :$!";
    select TMP;
    &print_segment( $site, $window, 10, @name );
    print "//\n";
    close TMP;

    $mark = &significance;
    for $i ( $site - $site + $window - 1 ) {
        last if $i > $len;
        $Sig[ $i ] eq '*' and next;
        $mark ne ' ' and $Sig[ $i ] = $mark;
    }
}
$seq{ $SigName } = join( '', @Sig );
}

sub chklen {
    $len = length $seq{ $name[0] };
    foreach $name ( @name[ 1 - $#name ] ) {
        if ( length ( $seq{ $name } ) != $len ) {
            select STDERR;
            print "Size mismatch. ( $len )\n";
            foreach $name ( @name ) {
                printf " %-${namelen}s : %5d sites ", $name, length( $seq{ $name } );
                print '*' if $len != length $seq{ $name };
                print "\n";
            }
            select STDERR;
            &print_seq;
            select STDOUT;
            return 0;
        }
    }
    $len;
}

sub print_seq {
    local( @name ) = @_;

    print @comments;

    $len = length $seq{ $name[0] };
    for ( $i = 0; $i < $len; $i += $width ) {
        &print_segment( $i, $width, $block, @name );
        print "\n";
    }
    print "//\n";
}

sub print_segment {
    my ( $site, $size, $block, @name ) = @_;
    my ( $seq );
    my ( $name1 );

```

```
foreach $name ( @name ) {  
    if ( $block > 0 ) {  
        ( $seq = substr( $seq{ $name }, $site, $size ) ) =~ s/(.{ $block })/ $1/g;  
    } else {  
        ( $seq = substr( $seq{ $name }, $site, $size ) );  
    }  
    $namel = $name;  
    length $namel > $namelen and  
        substr( $namel, $namelen - 1 ) = substr( $name, -1 );  
    printf "%-${namelen}s $seq\n", $namel;  
}  
  
sub significance {  
    my( $Significance ) = ' '  
  
    open DIST, "$dist $tmpfile|" or die "Can't exec $dist: $!";  
#   $/ = "\n";  
    while ( <DIST> ) {  
        /^#{20}/ and do {  
            @_ = split;  
            ( $entl, $ent2 ) = @[ 1, 3 ];  
        };  
        /^Synonymous/ and chop( $Reading = $_ );  
        /^Nonsynonymous/ and chop( $Reading = $_ );  
        /^Gamma distance/ and chop ( $Reading = $_ );  
  
        /^\td = / and do {  
            @_ = split ' ', $*;  
            ( $ds, $ss ) = @[ 0, 2 ] if $Reading eq 'Synonymous';  
            ( $dn, $sn ) = @[ 0, 2 ] if $Reading eq 'Nonsynonymous';  
            if ( $Reading eq 'Gamma distance' ) {  
                print STDERR "# ds=$ds, dn=$dn\n",  
                    next if $ds > 1.0 || $dn > 1.0;  
            }  
#           ( $dg, $sg )= @[ 0, 2 ];  
  
            $vs = $ss ** 2;  
            $vn = $sn ** 2;  
#           $vg = $sg ** 2;  
  
            $test = 0;  
            $test = ( $dn - $ds ) / sqrt( $vs + $vn ) unless $vs + $vn == 0;  
            ( $Significance = '*' ), last if $test > 2.33; # 1% level  
            $Significance = '+' if $test > 1.65; # 5% level  
        }  
    };  
}  
close;  
$Significance;  
}  
  
$STDOUT_TOP = <<EOF;  
format STDOUT TOP =  
Entry1      Entry2          Site Synonymous     Nonsynonymous     Significance  
-----+-----+-----+-----+-----+-----+-----  
.  
  
format STDOUT =  
@<<<<<<<<<<<<<<<<<<<< @<<<<<<<<<<<<<<<<<<<< @<<< @.### +- @.### @.### +- @.###  
@| | | | | | | |
```

```
$ent1, $ent2, $site, $ds, $vs, $dn, $vn, $Significance
.
EOF

sub init {
    $SIG{'INT'} = 'handler';
    $SIG{'QUIT'} = 'handler';
    $SIG{__WARN__} = 'handler';
    $SIG{__DIE__} = 'handler';
}

sub handler {
    my ( $sig ) = @_ ;
    print STDERR "Break by SIG$sig\n";
    unlink $tmpfile;
}
```


reduce (Perl): remove excess gaps and redundant sequences within alignments.

```
#!/usr/bin/perl
# reduce - reduce overlapped entry from alignment

$debug = 'y';
main_loop:
while (<>){
    /^\\s/ && next;
    /^[#>]/ && next;
    if (m!//!){
        $debug && print STDERR ++$cnt,":$sent[0]";
        &output;
        $debug && print STDERR " - done.\\n";
        undef %seq;
        @ent = ();
    }else{
        $debug && print STDERR ++$lines,"\\r";
        ($sent,$seq) = split(/\\s/, $_, 2);
        $seq =~ s/\\s//g;
        $seq{$sent} .= $seq;
        for (@ent){ next main_loop if $_ eq $sent; }
        push(@ent,$sent);
    }
}

&output if @ent;

sub output{

    $debug && print STDERR "\\r$sent[0]:Removing gap.";

    # remove gap
    $n = 0;
    rmgap_loop:
    while ($n < length($seq{$sent[0]})){
        for (@ent){
            $tmp = substr($seq{$_},$n,1);
            $n ++, next rmgap_loop if $tmp =~ /[a-zA-Z]/;
        }
        for (@ent){
            substr($seq{$_},$n,3) = '';
        }
    }

    # insert space between every codons.
    # $debug && print STDERR "\\r$sent[0]:Inserting space.";
    # for (@ent){
    #   while ($seq{$_} =~ s/(\\S\\S\\S)(\\S)/\\1 \\2/){};
    # }

    # remove overlapped sequence
    $debug && print STDERR "\\r$sent[0]:Removing overlapped sequence.";
    reduction:
    for $n (0-$#ent){
        for (0-$n-1){
            ($seq{$sent[$n]} = ''), next reduction if $seq{$sent[$_]} eq
            $seq{$sent[$n]};
        }
    }

    # print alignment
    $debug && print STDERR "\\r$sent[0]:Printing alignmnt.";
    for (;){
```

```
for $ent (@ent){
  next if $seq{$ent} eq '';
  printf "%-14s",$ent;
  for (0-14) {
    last unless $seq{$ent};
    print ' ',substr($seq{$ent},0, 3);
    $seq{$ent} = substr($seq{$ent},3);
  }
  print "\n";
}
last unless length($seq{$ent[0]}) > 0;
print "\n";
}

print "//\n";
}
```

reformat (Perl) - reformat alignment.

```
#!/usr/bin/perl
# reformat - reformat alignment

#$debug = 'y';
$width{'AA'} = 60;
$width{'NT'} = 45;
$NAMELEN = 14;

while ( ( $_ = $ARGV[ 0 ] ), /^-/ ) {
    shift;
    /^-w/ && ( $width = $' );      # print width.
    /^-a/ && ( $NUCF = 'n' );      # amino acid sequence.
    /^-d/ && ( $debug = '1' );     # debug mode
    /^-h/ && &usage;               # print the usage and quit.
    /^-l/ && do {                  # output to printer.
        $output = 'lp';
        $width = 150;
    };
    /^-m/ && ( $MARK = 'y' );      # Mark site number.
    /^-n/ && ( $NUCF = 'y' );      # nucleotide sequence spaced every 3 letters.
    /^-oden/ && ( $NAMELEN = 10 ); # oden permits only 10 chars for name.
}

main:
while ( <> ) {
    /^\\s/ && next main;
    ( /^>/ || s/^#/>#/ ) && do { print; next main; };
    m!^/*! && do {
        if ( ($NUCF eq 'y') || ( $TYPE eq 'NT' ) ) {
            &PrintOut('NT');
        } else {
            &PrintOut('AA');
        }
        &Clear;
        next main;
    };

    ( $name, $seq ) = split( ' ', $_, 2 );
    $seq =~ s/\\s//g;
    $seq{ $name } .= $seq;
    push( @name, $name ), $name{ $name }++ unless $name{ $name };
}

sub Clear {
    $debug && print STDERR "&Clear\\n";
    foreach ( @name ) {
        delete $name{ $_ };
        delete $seq{ $_ };
    }
    @name = ();
}

sub usage {
    print STDERR <<"EOF";
    Usage: $0 [-w aa_width] [-W nt_width] [-n(ucleotide)] [-x(no`numbering)]
    EOF
    exit -1;
}

sub TYPE {
    $debug && print STDERR "&TYPE\\n";
}
```

```

foreach $seq ( values seq ) {
  ( $seq =~ /[BDEFHIKLMNOPQSTVW]/ ) && return 'AA';
}
'NT';
}

sub PrintOut {
  local($TYPE) = @_ ;
  $width = $width{$TYPE} unless $width;
  $debug && print STDERR "&PrintOut($TYPE)\n";
  $cycle = 0;

  loop: {
    if ( $MARK ) {
      print ' ' x ($NAMELEN+1); # columns for entryname;

      $i = 0;
      $step = 40;
      $step = 10 if $TYPE = 'AA';
      while ( $i < $width && $i < length($seq{$name}) - $step ) {
        printf "%${step}d", $cycle + ( $i += $step );
      }
      print "\n";
    }

    for $name ( @name ) {
      printf "%-${NAMELEN}.${NAMELEN}s ", $name;
      $seq = substr( $seq{ $name }, 0, $width );
      $seq =~ s/(\S\S\S)/ \1/g if $TYPE eq 'NT';
      print "$seq\n";
      $seq{ $name } = substr( $seq{ $name }, $width );
    }
    $cycle += $width;
    print( "\n" ), redo loop if $seq{ $name[ 0 ] };
  }
  print "//\n";
}

```

```

separate (Perl) - separate alignment into individual sequences.
#!/data/bin/perl
#!/usr/local/bin/perl
#!/usr/bin/perl
#seperate - seperates alignment into each cds sequence.

$len = 18;

while ($_ = $ARGV[0], /^-/){
    shift;
    last if /^--/;
    /^-1/ && do {$len = $ARGV[0]; shift;};
}

# concatenate the sequences
main_loop:
while (<){
    /^\\s/ && next;
    /^[#>]/ && next;
    if (m!//!){
#       print "> $ent[0]\\n";
        &output;
        @ent = ();
        undef %seq;
        $n = $N;
    }else{
        ($ent,$seq) = split(/\\s/, $_, 2);
        $seq =~ s/\\s|[^A-Za-z]//g;
        $seq{$ent} .= $seq;
        for (@ent){ next main_loop if $_ eq $ent;}
        push(@ent,$ent);
    }
}

&output if @ent;

sub output{
    for $ent (@ent) {
        print ">$ent";
        loop:
        while ($seq{$ent}){
            printf "\\n%3.3s", $seq{$ent};
            $seq{$ent} = substr($seq{$ent}, 3);
            for (2 - $len){
                last loop unless $seq{$ent};
                printf " %3.3s", $seq{$ent};
                $seq{$ent} = substr($seq{$ent}, 3);
            }
        }
        print "\\n//\\n";
    }
}

```

```

seq (Perl): reformat sequence file.
#!/usr/bin/perl
#!/data/bin/perl
# len - get sequences in specified length.

# main routine appears last.

require 5; # perl version 5 or later.

$NAMELEN = 10;

# reverse complement definition.
%RevComp = qw(
    a, t,  t, a,  c, g,  g, c,
    r, y,  y, r,  m, k,  k, m,
    s, w,  w, s,  h, d,  d, h,  b, v,  v, b,
    n, n
);

# Genetic Code Table
%AA = qw(
    ttt F tct S tat Y tgt C
    ttc F tcc S tac Y tgc C
    tta L tca S taa * tga *
    ttg L tcg S tag * tgg W
    ctt L cct P cat H cgt R
    ctc L ccc P cac H cgc R
    cta L cca P caa Q cga R
    ctg L ccg P cag Q cgg R
    att I act T aat N agt S
    atc I acc T aac N agc S
    ata I aca T aaa K aga R
    atg M acg T aag K agg R
    gtt V gct A gat D ggt G
    gtc V gcc A gac D ggc G
    gta V gca A gaa E gga G
    gtg V gcg A gag E ggg G
);

# Reverse Genetic Code Table
%Codon = qw(
    A gc.                M atg
    C tg[tcy]           N aa[tcy]
    D ga[tcy]           P cc.
    E ga[agr]           Q ca[agr]
    F tt[tcy]           R (cg.|ag[agr])
    G gg.                S (tc.|ag[tcy])
    H ca[tcy]           T ac.
    I at[tcayh]         V gt.
    K aa[agr]           W tgg
    L (tt[agr]|ct.) Y ta[tcy]
);

# subroutines

sub ReadSeqGB {
    $SEQ = '';
    while (<){
        m|^//| && return;
        s/[ 0-9]//g;
        $SEQ .= $_;
    }
}

```



```

sub ReadSeqEMBL {
    $_ = <>;
    ($SEQ = $_) =~ s/[ 0-9/]+//g;
}

sub ReadODEN {
    $NAME = '';
    $SEQ = '';
    while (<>){
        m|^//| && last;
        /^>[^\dA-Za-z]/ and print, next;
        /^>(.{1,$NAMELEN})(.*)$/ and do {
            if ( $NAME ne '' ) {
                $NAME1 .= $2;
            }else{
                chop;
                $NAME = $1;
                $NAME1 = $2;
            }
            next;
        };
        $SEQ .= $_;
    }
    $SEQ =~ s/[ \s\d]+//g;
    return length $SEQ;
}

sub PrintSeq {
    local ($name, $seq) = @_;
    print ">$name\n" if $name;
    while ($seq){
        print substr($seq, 0, 60), "\n";
        $seq = substr($seq, 60);
    }
    print "//\n";
}

sub ReverseComplement {
    my @seq = split('',$_[0]);
    my $newseq = '';
    for (reverse @seq) {
        $newseq .= $RevComp{$_};
    }
    $newseq;
}

sub Translate {
    my ($nuc) = join('',$_);
    my ($aa) = '';

    while ($nuc) {
        $_ = $AA{ substr($nuc, 0, 3) };
        $aa .= $_ unless $_ eq '';
        $aa .= '?' if $_ eq '';
        $nuc = substr($nuc, 3);
    }
    $aa;
}

sub BackTranslate {
    my @aa = split('',$join('',$_));
    my $NucRE = '';

```

```

foreach (@aa) {
    ($NucRE .= $Codon{$_}), next unless $Codon{$_} eq '';
    $NucRE .= $_ if /[0-9.,*+(){}|\[\]\]/;
}
$NucRE;
}

# main.

while ( $_ = $ARGV[0], /^-/ ) {
    shift;
    if ( /^-u/ ) {
        $toupper = 'y';
        undef $tolower if defined $tolower;
    }
    if ( /^-l/ ) {
        $tolower = 'y';
        undef $toupper if defined $toupper;
    }
    /^-\d/ and $NAMELEN = - $_;
}

while ( $len = ReadODEN ) {
    defined $toupper and $SEQ =~ y/a-z/A-Z/;
    defined $tolower and $SEQ =~ y/A-Z/a-z/;
    PrintSeq ($NAME . $NAME1, $SEQ);
}

```

sgrep (Perl): get regular expression from sequence file and print. Also finds nucleotide sequence corresponding amino acid sequence.

```
#!/usr/bin/perl
#!/data/bin/perl
# sgrep - get sequence from file.

# main routine appears last.

require 5; # perl version 5 or later.

# reverse complement definition.
%RevComp = qw(
    a, t,  t, a,  c, g,  g, c,
    r, y,  y, r,  m, k,  k, m,
    s, w,  w, s,  h, d,  d, h,  b, v,  v, b,
    n, n
);

# Genetic Code Table
%AA = qw(
    ttt F tct S tat Y tgt C
    ttc F tcc S tac Y tgc C
    tta L tca S taa * tga *
    ttg L tcg S tag * tgg W
    ctt L cct P cat H cgt R
    ctc L ccc P cac H cgc R
    cta L cca P caa Q cga R
    ctg L ccg P cag Q cgg R
    att I act T aat N agt S
    atc I acc T aac N agc S
    ata I aca T aaa K aga R
    atg M acg T aag K agg R
    gtt V gct A gat D ggt G
    gtc V gcc A gac D ggc G
    gta V gca A gaa E gga G
    gtg V gcg A gag E ggg G
);

# Reverse Genetic Code Table
%Codon = qw(
    A gc.                M atg
    C tg[tcy]            N aa[tcy]
    D ga[tcy]            P cc.
    E ga[agr]            Q ca[agr]
    F tt[tcy]            R (cg.|ag[agr])
    G gg.                S (tc.|ag[tcy])
    H ca[tcy]            T ac.
    I at[tcayh]          V gt.
    K aa[agr]            W tgg
    L (tt[agr]|ct.) Y ta[tcy]
);

# subroutines

sub ReadSeqGB {
    $SEQ = '';
    while (<>){
        m|^//| && return;
        chop;
        s/[\s0-9]//g;
        $SEQ .= $_;
    }
}
```

```

sub ReadSeqEMBL {
    $_ = <>;
    ($SEQ = $_) =~ s/[ 0-9/]|lg;
}

sub PrintSeq {
    local ($name, $seq) = @_;
    print ">$name\n";
    while ($seq){
        print substr($seq, 0, 60), "\n";
        $seq = substr($seq, 60);
    }
    print "//\n";
}

sub ReverseComplement {
    my @seq = split('',$_[0]);
    my $newseq = '';
    for (reverse @seq) {
        $newseq .= $RevComp{$_};
    }
    $newseq;
}

sub Translate {
    my ($nuc) = join('',$_);
    my ($aa) = '';

    while ($nuc) {
        $_ = $AA{ substr($nuc, 0, 3) };
        $aa .= $_ unless $_ eq '';
        $aa .= '?' if $_ eq '';
        $nuc = substr($nuc, 3);
    }
    $aa;
}

sub BackTranslate {
    my @aa = split('',$join('',$_));
    my $NucRE = '';
    foreach (@aa) {
        ($NucRE .= $Codon{$_}), next unless $Codon{$_} eq '';
        $NucRE .= '(-.)' if $_ eq '.';
        $NucRE .= $_ if /[0-9,*+(){}|\[\]]/;
    }
    $NucRE;
}

# main.

if ( @ARGV < 1 ) {
    print STDERR "Usage: $0 REstring [file -]\n";
    exit 1;
}

print ">RE : $ARGV[0]\n";
$PATTERN = BackTranslate shift @ARGV;
print ">TRE: $PATTERN\n";

while (<>){
    /^LOCUS|^ID / && do {
        @_ = split;
    }
}

```

```

$ENTRY = $_[1];
next;
};
/^AC/ && do {
    @_ = split;
    ($ACCESSION = $_[1]) =~ s/;$//;
};
/^ORIGIN/ && do {
    ReadSeqGB;
    while ($SEQ =~ /$PATTERN/go){
        print "$ENTRY\t[" , pos $SEQ, "]:\t", Translate($&), "\n";
    }
};
/^SQ / && do {
    ReadSeqEMBL;
    while ($SEQ =~ /$PATTERN/go){
        print "$ENTRY\t[" , pos $SEQ, "]:\t", Translate($&), "\n";
    }
};
}

```

```
SORT (Perl) - sort records.  
#!/usr/bin/perl  
# SORT - sort record  
  
$/ = "//\n";  
  
@Records = <>;  
print sort @Records;
```



```

wal (Perl) - widow analysis.
#!/usr/bin/perl
#!/usr/local/bin/perl
#!/data/bin/perl
#wa - window analysis for synonymous and nonsynonymous substitutions.

$window=60; # window size have to be lower than 150
$shift=3; # window shift size
$N = $n=1;

$# = '%.4g';
$, = "\t";

while ($_ = $ARGV[0], /^-/){
    shift;
    last if /^--/;
    /^-C/ && ($check = 'y'); # print distance of each pairs.
    /^-a/ && ($schk = 'y'); # print sliding window alignments.
    /^-c/ && ($n = $N = $window / 2); # center site indication
    /^-w/ && do { $window = $ARGV[0]; shift; }; # change window size.
    /^-s/ && do { $shift = $ARGV[0]; shift; }; # change shift size.
}
if ($shift %3 != 0) { die "Improper shift size ($shift).";}
if ($window %3 !=0) { die "Improper window size ($window).";}

$FILT="distng";
$FILT="distNG" if `hostname` eq "niguxp\n";
$TMP="$0.$$";

# concatenate the sequences
main_loop:
while (<){
    /^\\s/ && next;
    /^[#>]/ && next;
    if (m!//!){
        print "> $ent[0]\n";
        &output;
        print "//\n";
        @ent = ();
        undef %seq;
        $n = $N;
    }else{
        ($ent,$seq) = split(/\\s/, $_, 2);
        $seq =~ s/\\s//g;
        $seq{$ent} .= $seq;
        for (@ent){ next main_loop if $_ eq $ent; }
        push(@ent,$ent);
    }
}

&output if %seq;
exit;

sub output{
    # output sliding windows into temporary file.
    open(TMP, ">$TMP") || die;
    select(TMP);

loop:
for (;;){
    foreach (@ent){
        last loop if length($seq{$_}) < $window;
        printf "%-10s\t%s\n", $_, substr($seq{$_}, 0, $window);
    }
}

```

```

    $seq{$_} = substr($seq{$_}, $shift);
  }
  print "//\n";
}
close TMP;

# calculate ds and dn for all sliding windows.
open(TMP, "$FILT $TMP|") || die;
select(STDOUT);
print "site\tsyn.subs.\tnsy.subs.\n";
$c=0; $syn=$nsy=0.0;

while (<TMP>){
  next if /^[#>]//;
  unless
    ($n1,$n2,$syn_s,$syn_c,$syn_d,$nsy_s,$nsy_c,$nsy_d,$other)=split;
    $c++; $syn += $syn_d; $nsy += $nsy_d;
    push(@syn,$syn_d), push(@nsy,$nsy_d) if $check;
  }else{
    $syn /= $c; $nsy /= $c;
    print $n,$syn,$nsy,"\n" unless $check;
    print $n,$syn,$nsy,'ds>',$syn,'dn>',$nsy,"\n" if $check;
    $n += $shift;
    $c=0; $syn=$nsy=0.0; @syn=@nsy=();
  }
}
close TMP;
system "cat $TMP" if $schk;
unlink $TMP;
}

```

wpl (Perl) - plots the result of wal.

```
#!/usr/bin/perl
# wpl - Window Plot program ver 1.0

$N = '+'; # nonsynonymous dot
$S = '.'; # synonymous dot
$C = 'o'; # shared by synonymous and nonsynonymous dot

$^L = '';
#$debug = 'y';
$maxlen = 70;
$limit = 1.0;

while ( <> ){

    # read all data;
    m!^//! && do {
        &output;
        @site = @ds = @dn = ();
        $max = 0;
        next;
    };
    print, --$- if /^>/;
    next unless /\d/;
    ($site,$ds,$dn) = split;
    push (@site,$site);
    push (@ds, $ds);
    push (@dn, $dn);
    $max=$ds if ($ds < $limit) && ($max < $ds);
    $max=$dn if ($dn < $limit) && ($max < $dn);
}

&output if @site;
exit;

sub output{
    # print it graphically
    printf "%76.2f\n", $max;
    for ($[ - $#site){
        $cs = $maxlen * $ds[$_] / $max;
        $cn = $maxlen * $dn[$_] / $max;
        $cs = $maxlen+1 if $cs > $maxlen;
        $cn = $maxlen+1 if $cn > $maxlen;

        if ($cs > $cn) {
            $tmp = ' ' x ($cn-1). $N . ' ' x ($cs-$cn). $S;
        }elseif ($cs < $cn){
            $tmp = ' ' x ($cs-1). $S . ' ' x ($cn-$cs). $N;
        }else{
            $tmp = ' ' x ($cs-1). $C;
        }
        $site = '';
        $site = $site[$_] if $site[$_] %10 == 0;
        write;
    }
    print "//\n";
}

#format STDOUT_TOP =
#
0@#####.##
#      $max
```

