ABSTRACT

ALGORITHMS AND TOOLS FOR SPLICING JUNCTIONS DONOR RECOGNITION IN GENOMIC DNA SEQUENCES

by
Maisheng Yin

The consensus sequences at splicing junctions in genomic DNA are required for pre-mRNA breaking and rejoining which must be carried out precisely. Programs currently available for identification or prediction of transcribed sequences from within genomic DNA are far from being powerful enough to elucidate genomic structure completely[4]. In this research, we develop a degenerate pattern match algorithm for 5' splicing site (Donor Site) recognition.. Using the Motif models we developed, we can mine out the degenerate pattern information from the consensus splicing junction sequences. Our experimental results show that, this algorithm can correctly recognize 93% of the total donor sites at the right positions in the test DNA group. And more than 91% of the donor sites the algorithm predicted are correct. These precision rates are higher than the best existing donor classification algorithm[25]. This research made a very important progress toward our full gene structure detection algorithm development.

ALGORITHMS AND TOOLS
FOR SPLICING JUNCTION DONOR RECOGNITION
IN GENOMIC DNA SEQUENCES

by
Maisheng Yin

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirement for the Degree of
Master of Science in Computer Science

Department of Computer and Information Science

October 1997

APPROVAL PAGE

ALGORITHMS AND TOOLS
FOR SPLICING JUNCTION DONOR RECOGNITION
IN GENOMIC DNA SEQUENCES

Maisheng Yin

Dr. Jason T. L. Wang, Thesis Advisor Date
Associate Professor of Computer and Information Science, NJIT

Dr. James McHugh, Committee Member Date
Professor of Computer and Information Science, NJIT

Dr. David Nassimi, Committee Member Date
Associate Professor of Computer and Information Science, NJIT

BIOGAPHICAL SKETCH

Author: Maisheng Yin

Degree: Master of Science in Computer Science

Date: October 1997

Date of Birth:

Place of Birth:

Undergraduate and Graduate Education:

- Master of Science in Computer Science,
  New Jersey Institute of Technology, Newark, NJ, 1997

- Master of Science in Microbiology,
  Central China Agriculture University, Wuhan, Hubei, China, 1985

- Bachelor of Science in Microbiology,
  Central China Agriculture University, Wuhan, Hubei, China, 1982

Major: Computer Science

This thesis is dedicated to
my wonderful wife, Yuxian,
my children, Zhigang and Sally
for their love, support,
and patience while I worked nights and weekends
on this project instead of spending time with them.

ACKNOWLEDGMENT

The author wishes to express his sincere gratitude to his supervisor, Professor Jason T. L. Wang, for his guidance, encouragement and support throughout this research. Special thanks to Professors James McHugh and David Nassimi for serving as members of the committee.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Biology Background

Genes are the invisible information-containing elements that are distributed to each daughter cell when a cell divides and Genes are made of deoxyribonucleic acid (DNA). A DNA chain is a long unbranched polymer composed of only four types of subunits. These are the deoxyribonucleotides containing the bases adenine (A), cytosine (C), guanine (G), and thymine (T) (see Figure 1).The genetic information in the DNA is copied into RNA through a process known as DNA transcription. RNA transcripts that direct the synthesis of protein molecules are called messenger RNA (or mRNA). mRNA then passes the information into protein during protein synthesis [1].

The basic gene structure components for higher eucaryotes include the promoter, start codon, introns, exons, stop codon and poly-A adding site, etc. Figure 1 shows the human β-globin gene sequence with exon and intron regions[1]. The intron sequences will be removed from mRNA precursors (pre-mRNAs) by the RNA splicing mechanism (see Figure 2).

Introns sequence in genes have no function at all and are actually the generic "junk" [1]. Introns range in size from about 80 nucleotides to 10,000 nucleotides or more. They differ dramatically from exons in that their exact nucleotide sequences seem to be unimportant. The only highly conserved sequences in introns are those required for intron removal. Thus there are consensus sequences at each end of an intron that are nearly the same in all known intron sequences, and these can not be changed without affecting the

```
           CCCTGTGGAGCCACACCCTAGGGTTGGCCAATCTACTCCCAGGAGCAGG
           GAGGGCAGGAGCCAGGGCTGGGCATAAAAGTCAGGGCAGAGCCATCTAT
           TGCTTACATTTGCTTCTGACACAACTGTGTTCACTAGCAACTCAAACAG
Exon 1     ACACCATGGTGCACCTGACTCCTGAGGAGAAGTCTGCCGTTACTGCCCT
           GTGGGCAAGGTGAACGTGGATGAAGTTGGTGGTGAGGCCCTGGGCAGGT
           TGGTATCAAGGTTACAAGACAGGTTTAAGGAGACCAATAGAAACTGGGC
Intron 1   ATGTGGAGACAGAGAAGACTCTTGGGTTTCTGATAGGCACTGACTCTCT
           CTGCCTATTGGTCTATTTTCCCACCCTTAGGCTGCTGGTGGTCTACCCT
           TGGACCCAGAGGTTCTTTGAGTCCTTTGGGGATCTGTCCACTCCTGATG
Exon 2     CTGTATGGGCAACCCTAAGGTGAAGGCTCATGGCAAGAAAGTGCTCGGT
           GCCTTTAGTGATGGCCTGGCTCACCTGGACAACCTCAAGGGCACCTTTG
           CCACACTGAGTGAGCTGCACTGTGACAAGCTGCACGTGGATCCTGAGAA
           CTTCAGGGTGAGTCTATGGGACCCTTGATGTTTTCTTTCCCCTTCTTTT
           CTATGGTTAAGTTCATGTCATAGGAAGGGGAGAAGTAACAGGGTACAGT
           TTAGAATGGGAAACAGACGAATGATTGCATCAGTGTGGAAGTCTCAGGA
           TCGTTTTAGTTTCTTTTATTTGCTGTTCATAACAATTGTTTTCTTTTGTT
           TAATTCTTGCTTTCTTTTTTTTTCTTCTCCGCAATTTTTACTATTATAC
           TTAATGCCTTAACATTGTGTATAACAAAAGGAAATATCTCTGAGATACA
           TTAAGTAACTTAAAAAAAAACTTTACACAGTCTGCCTAGTACATTACTA
           TTTGGAATATATGTGTGCTTATTTGCATATTCATAATCTCCCTACTTTA
Intron 2   TTTTCTTTTATTTTTAATTGATACATAATCATTATACATATTTATGGGT
           TAAAGTGTAATGTTTTAATATGTGTACACATATTGACCAAATCAGGGTA
           ATTTTGCATTTGTAATTTTAAAAAATGCTTTCTTCTTTTAATATACTT
           TTTTGTTTATCTTATTTCTAATACTTTCCCTAATCTCTTTCTTTCAGGG
           CAATAATGATACAATGTATCATGCCTCTTTGCACCATTCTAAAGAATAA
           CAGTGATAATTTCTGGGTTAAGGCAATAGCAATATTTCTGCATATAAAT
           ATTTCTGCATATAAATTGTAACTGATGTAAGAGGTTTCATATTGCTAAT
           AGCAGCTACAATCCAGCTACCATTCTGCTTTTATTTTATGGTTGGGATA
           AGGCTGGATTATTCTGAGTCCAAGCTAGGCCCTTTTGCTAATCATGTTC
           ATACCTCTATCTTCCTCCCACAGCTCCTGGGCAACGTGCTGGTCTGTGT
           GCTGGCCCATCACTTTGGCAAAGAATTCACCCCACCAGTGCAGGCTGCCT
Exon 3     ATCAGAAAGTGGTGGCTGGTGTGGCTAATGCCCTGGCCCACAAGTATCA
           CTAAGCTCGCTTTCTTGCTGTCCAATTTCTATTAAAGGTTCCTTTGTTC
           CCTAAGTCCAACTACTAAACTGGGGGATATTATGAAGGGCCTTGAGCAT
           CTGGATTCTGCCTAATAAAAAACATTTATTTTCATTGCAATGATGTATT
           TAAATTATTTCTGAATATTTTACTAAAAAGGGAATGTGGGAGGTCAGTG
           CATTTAAAACATAAAGAAATGATGAGCTGTTCAAACCTTGGGAAAATAC
           ACTATATCTTAAACTCCATGAAAGAAGGTGAGGCTGCAACCAGCTAATG
           CACATTGGCAACAGCCCCTGATGCCTAATGCACATTGGCAACAGCCCCT
           GATGCCTATGCCTTATTCATCCCTCAGAAAAGGATTCTTGTAGAGGCTT
           GATTTGCAGGTTAAAGTTTTGCTATGCTGTATTTTACATTACTTATTGT
           TTTAGCTGTCCTCATGAATGTCTTTTC
```

Figure 1. The transcribed portion of the human β-globin gene. The sequence of the DNA strand corresponding to the mRNA sequence is given with the three coding regions (exons) are red colored. The donor site consensus sequences are under lined and the highly conserved GT (in donor sites) and AG (in acceptor sites) nucleotides for the splicing junction sites are blue colored.

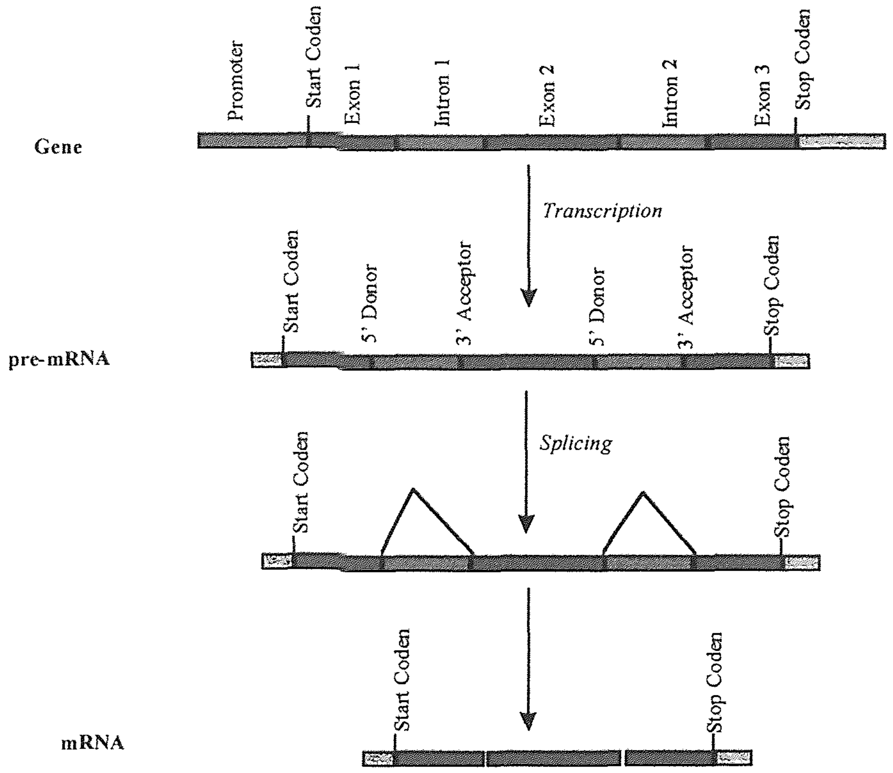

Figure 2. Gene structure and mRNA splicing.

splicing process. The conserved boundary sequences at the 5' splice site is called donor site, and the one at the 3' splice site is called acceptor site. The consensus sequences for donor and acceptor sites are shown in Figure 3. The RNA breaking and rejoining (splicing) must be carried out precisely because an error of even one nucleotide would shift the reading frame in the resulting mRNA molecule and make nonsense of its message[1].

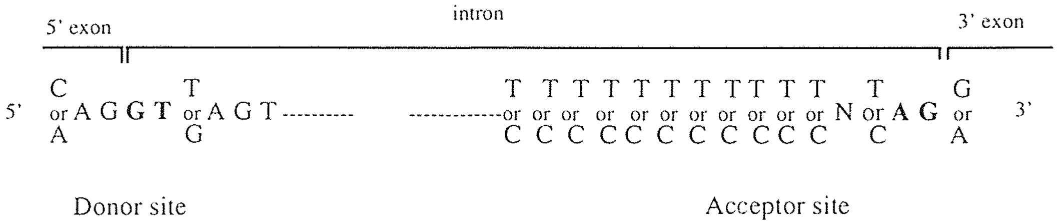
5' exon
intron
3' exon
5'
C or A A G G T T or G A G T ---------
------------ T or C T or C T or C T or C T or C T or C T or C T or C T or C T or C T or C N T or C A G G or A
3'
Donor site
Acceptor site

Figure 3. Consensus sequences for the 5' and 3' splice sites used in RNA splicing. For simplicity, 'T' is used for 'U'.

For biologists, now it is easy to get genomic sequences by sequencing the genomic DNA. But identification or prediction of transcribed sequences from within genomic regions has been a major rate-limiting step in the pursuit of genes. The bioinformatics approach for gene detection means using computer programs to predict exon regions. From Figure 1 and 2 we can see that a gene may have several exons interrupted by introns. To find out exon regions, we have to find out the start codon (ATG), donor sites, acceptor sites and the stop codon. For the recognition of splicing junction sites, one can not just do the simple consensus search for the donor and acceptor sites, because both these sites are of degenerate form and some of the nucleotides in these sites can be changed without affecting the splicing signal[1, 2, 3]. So the simple consensus analysis provides only a very rough functional mapping of a sequence, and its results should be interpreted with due caution [4]. It is also impossible for the consensus sequence to account relative importance of each of the nucleotides in the sequence[5, 6].

Recently, the problems of gene identification and gene structure prediction in higher eukaryotic genomic DNA sequences by computational analysis have been received wide attention. As the Human Genome Project turns from mapping to large-scale sequencing, the need for efficient and accurate methods for identifying the gene coding regions will continually increase[7, 8]. This research is our first step for a full gene structure detection program and concentrates on the 5' splicing site (donor site) recognition.

1.2 Current Status and Progress

Although methods to predict potential protein coding regions on genomic DNA sequences have existed since the 1980s, the first programs to assemble potential DNA coding regions into translatable mRNA sequences were not available until the early 1990s[7]. Recently there are several programs available for biologists, such as GeneID[9], GeneParser[10], GenLang[11] and GRAIL[12], etc. GRAIL is the one now widely used by researchers and it is available on the BLAST web site[1] for gene structure detection. The approaches have been used for the function sites detection include:

Consensus search[4]

Consider an aligned set of site sequences. If at each position with nonuniform distribution of nucleotides the researchers retain the preferred nucleotide and obtain the *consensus* word. It is possible to account for degeneracies and to distinguish between strongly and weakly conserved positions, dependent on the degree of the non-uniformity. People write the consensus of mammalian donor splicing sites as maG/**GT**RAGu, where boldface denotes invariant positions, capitals and lowercase letters denote, respectively, strongly and weakly conserved positions, "R" denotes 'A or G', 'm' denotes 'a or c', '/' is the splice point. A formal determination of conserved positions can be made using standard statistical criteria or computation of the information content of positional nucleotide distribution [13, 14, 15]. The consensus methods are tools to summarize the distribution of an aligned set of molecular sequences. Typically the methods make three simplifying assumptions: analysis of molecular sequences is a multistage process in which sequence alignment precedes the identification of consensus sequences, an alignment of the molecular sequences has already been obtained, and alignment of the identification of consensus sequences can be treated independently. Thus the problem to find a consensus of k aligned molecular sequences, in which n aligned positions have been identified, can be viewed as a set of n simpler problems, each to find a consensus of k symbols (i.e. nucleotide) at an aligned position[16]. The comparison with the consensus is the simplest form of a site prediction algorithm, but as mentioned previously, consensus analysis provides only a very rough functional mapping of a sequence and its results should be interpreted with caution[4].

Weight matrices[4]

The next level of sophistication is provided by weight matrices. Each nucleotide b (b = A, C, G, T) in the site position p (p = 1,2,..., L) is set in correspondence with the weight W(b, p). The score off a potential site is defined as the sum of the positional weights of the constituent nucleotides. R. Staden applied the weight matrix method to obtain the relative importance of each nucleotide in the consensus sequence [17]. Another approach used for multivariate statistical analysis was to perform categorical discriminant analysis, where nucleotide sequences were transformed into categorical data. Categorical weights on the variables were estimated in such a way that the two classes of the 5' splice site sequences and sequences other than 5' splice site might be discriminated most distinctly[5] . It has been demonstrated that site strengths estimated by this theory to some extent agree with the experimental data [4]. Like consensus search, the weight matrices can be used for fast database searches.

Pattern recognition and neural networks[4]

Algorithms of the pattern recognition theory are based on the (implicit) assumption that in the genome there is a tendency to avoid nonfunctioning signal-like sites. Thus, a learning sample consists of two classes, sites and nonsites, the latter class usually formed by random fragments of the natural DNA. The basic steps in application of pattern recognition techniques are as follows: (i) Creation of a learning sample. (ii) Choice and encoding of signal features. (iii) Iterative correction of recognition rules according to results of discrimination between the two classes at the previous round. (iv) Testing on an independent sample.

One of the diverse pattern recognition algorithms is *neural networks* [18]. The neural networks consists of a layer of input neurons, several layers of hidden neurons, and an output neuron. When the network is presented with a candidate site, the input neurons check whether the site possesses the corresponding features and send binary signals to the neurons of the first hidden layer. Each hidden neurons sums the weighted signals coming by connections from the lower level, compares the result with the threshold, and sends a binary signal to the upper level neurons. The output neuron provides the final site / nonsite decision.

Programs such as GenViewer[19] and GRAIL[12] employs a procedure that scores candidate exons using some combination of the sites scores and the coding potential, and then performs an exhaustive search over the set of structures generated by the remaining high-scoring exons. Recently, Moises Burset and Roderic Guigo evaluated a number of computer programs designed to predict the structure of protein coding genes in genomic DNA sequences[7]. The programs analyzed were uniformly tested on a large set of vertebrate sequences with simple gene structure. Their carefully selected test set included 570 sequences, totaling 2649 coding exons. All the sequences in the test set had the start codon and stop codon. All the donor sites contains the GT dinucleotide and all the acceptor sites contains the AT dinucleotide at the right positions. Some of their data was shown in Table 1. The results indicated that the predictive accuracy of the program analyzed was really low. For example, for the widely used GRAIL, the sensitivity (Sn) and specificity (Sp) were just 36% and 43%[7].

Table 1. Performance of the programs evaluated by M. Burset and R. Guigo[7]

Programs	Sensitivity	Specificity
FGENEH	0.61	0.64
GeneID+	0.73	0.7
GeneParser3	0.56	0.58
GenLang	0.51	0.52
GRAIL2	0.36	0.43
SORFIND	0.42	0.47
Xpound	0.15	0.18

Note: Moises Burset and Roderic Guigo defines Sensitivity (Sn) and Specificity (Sp) as the follows:

$$S_n = \frac{\text{Number of Correct Exons}}{\text{Number of Actual Exons}}$$

$$S_p = \frac{\text{Number of Correct Exons}}{\text{number of Predicted Exons}}$$

So they claimed that programs currently available may still be of great use in pinpointing the regions likely to contain exons, they are far from being powerful enough to elucidate its genomic structure completely[7].

Our research is targeted on developing more efficient and accurate methods for identifying gene structures. In this thesis, we concentrate on the detection of 5' splicing junction site, donor site, which is one of the most important components of gene structure. We first introduce a motif model to represent the degeneracy features of these splicing sites, then employ pattern match methods to classify / detect the splicing sites. Our results are very promising for donor sites recognition, which is the most important step for fully solving the gene detection problem. In the near future we are going to develop methods for detection / examination of translation initiation signal (Start codon), 3'-splicing site, translation potential, translation termination position, and *etc*. Combining all these together, we will develop a powerful program for elucidate gene structure.

CHAPTER 2

PRELIMINARIES

2.1 Term Definition

2.1.1 Exon[1]

Segment of a eukaryotic gene that consists of DNA coding for a sequence of nucleotides in mRNA; an exon can encode amino acids in a protein. An exon is usually adjacent to a noncoding DNA segment called an intron.

2.1.2 Intron[1]

Noncoding region of a eukaryotic gene that is transcribed into an RNA molecule but then excised by RNA splicing when mRNA is produced.

2.1.3 Splicing Junctions[1]

The only highly conserved sequences in introns for intron removal, which are found at or near the ends of an intron and are very similar in all known intron sequences. The splicing junctions generally cannot be altered without affecting the splicing process that normally removes the intron sequence from the primary RNA transcript.

2.1.4 Donor Site[1]

The conserved boundary sequences at the 5' splice sites. The conserved sequences include 9 nuclear bases with GT (GU in mRNA) almost invariable to all donor sites[1, 5, 20, 21, 22]. An example of donor site sequence looks like the following:

$$\text{CAG}\underline{\text{GT}}\text{GAGT}$$

The counterpart of a donor site is the 3' splice site, Acceptor site.

2.1.5 True Donor Site

A 9 base long authentic **donor site** sequence extracted from a real gene.

2.1.6 False Donor Site

A 9 base DNA sequence containing a GT dinucleotide that is extracted from any place, except the True Donor Sites, in a real gene. The GT dinucleotide in the False Donor Site is in the same position as it in a True Donor Site.

2.1.7 Donor Positive DNA Sequence

A region of a real DNA sequence in which there is least one **Donor Site** in it.

2.1.8 Donor Negative DNA Sequences

A region of a real DNA sequence in which there are no any **Donor Sites** in it.

2.2 The DNA Sequences

The DNA sequences used in this research were obtained by anonymous FTP from the site "*ftp.ics.uci.edu/pub/machine-learning-databases/molecular-biology*". All sequences in the group were 60 bases long. Among them were 767 sequences that contain donor sites (**Donor Positive DNA Sequence**) and 768 acceptor positive sequences. Rest of them (1655) were negative sequences (There were no donor or acceptor sites in the sequences). The following sequences are some examples from the test group and the bold type bases are the Donor Sites:

Donor Positive DNA Sequence:
EI,   ATRINS-DONOR-521,
CCAGCTGCATCACAGGAGGCCAGCGAG**CAGGTCTGT**TCCAAGGGCCTTCGAGCCAGTCTG
EI,   ATRINS-DONOR-905,
AGACCCGCCGGGAGGCGGAGGACCTGC**AGGGTGAGC**CCCACCGCCCCTCCGTGCCCCCGC
EI,   BABAPOE-DONOR-30,
GAGGTGAAGGACGTCCTTCCCCAGGAG**CCGGTGAGA**AGCGCAGTCGGGGGCACGGGGATG
EI,   BABAPOE-DONOR-867,
GGGCTGCGTTGCTGGTCACATTCCTGG**CAGGTATGG**GGCGGGGCTTGCTCGGTTTTCCCC

Donor Negative DNA Sequences:
N,   AGMKPNRSB-NEG-1,
CAAAAGAACAAAGCTGGAGGCATCACGCTACCTGACTTCAAACTATACTACAAGGCTACA
N,   AGMORS12A-NEG-181,
AGGGAGGTGTCTGATTGGTCCAGCTTAGTCCATGTCCCTACCCTGAACAGGGGCATGGGG
N,   AGMORS9A-NEG-481,
TGGTCAATTCTGAATTCTCTCCACATTATTATTATTATTTTTGAGACAGTCTTGCTCTG
N,   AGMRSKPNI-NEG-1141,
AGGGCATGGTGAAAAAGGAAATATCTTCCGTTCAAAACTGGAAATAAGCTTTCTGAGAAA

2.3 Donor Site Groups Construction

The Donor Site Learning Group and the Donor Site Positive Training Group were built up using the **True Donor Site** extraction algorithm shown in Figure 4. There were 250 True Donor Sites in the **Donor Site Learning Group** and they were extracted from the first 250 **Donor Positive DNA Sequences** (see section 2.1). In the **Donor Site Positive Training Group**, there were 250 **True Donor Sites**. These true donor sites were extracted from the second 250 sequences following the first 250 sequences in the **Donor Positive DNA Sequences**.

There were 800 **False Donor Sites** in the **Donor Site Negative Training Group**. As defined in section 2.1, all entries in this group were 9 base long and all contained the

Input: Donor Positive DNA sequences and the splicing position.

Output: True Donor Site sequences.

```
for each inputted sequences Seq do
    begin
        {let m be the index  of donor D}
        m :=  0;
        {let i  be  the splicing position}
        for  j :=  i -3 to j := i + 5  do
            begin
                D[m] := Seq.[j];
                 m := m + 1
            end
    end;
```

Figure 4. Algorithm for extract True Donor Site sequences from donor positive DNA sequence.

'GT' dinucleotide in the same position as in the true donor sites. The false donor sites were extracted from 270 **Donor Negative DNA Sequences** as described in section 2.1 using the algorithm showing in Figure 5.

There 692 candidate donor site sequences in the **Candidate Donor Site Group** with 216 true donor sites and 474 false donor sites. They were extracted from the last 216 DNA sequences in the **Donor Positive DNA Sequences** as described in section 2.1 using the same algorithm shown in Figure 5 but the input sequences were donor site positive DNA sequences.

2.4 Measures of Performance Accuracy

In this research, we measured the performance accuracy of the program at two different levels: donor site classification and donor site detection. For the donor site classification, we input a group of candidate donor site sequences (see section 2.3) into the program and let the program classify whether a candidate is a true donor site or false donor site sequence. This is the way we used to test our algorithms in our developing cycles. The donor classification algorithm can also be used to classify a segment of DNA sequence to see whether it is a donor positive or donor negative sequence as reported by Jason T. L. Wang and his colleagues [23]. This is useful for finding out some local information for a giving gene region. For the donor site detection, we input DNA sequences directly into the program and let the program to recognize the real donor sites in the genes. This is very important for the full gene structure detection application which we will develop in the near future.

```
Input: Donor Negative DNA sequences.
Output: False Donor Site sequences.

for each inputted sequences Seq do
   begin
      { let m be the index of False Donor D }
     m :=  0;
     {let len be the length of Seq. }
     for  i := 3 to i := len - 5  do
        begin
           if Seq.[i] = 'G' and Seq.[i + 1] = 'T'
              for j := i - 3 to j := i + 5 do
                 begin
                    D[m] := Seq.[j];
                    m := m + 1
                 end
        end
   end;
```

Figure 5. Algorithm for extract False Donor Site sequences from Donor Negative DNA sequences.

2.4.1 Donor Site Classification

At this level, we measured the performance accuracy on a group of candidate donor sites for Sensitivity and Specificity[7]. To compute sensitivity and specificity, we use the following formulas:

Sensitivity:

$$S_n = \frac{\text{Correct True Donor Sites Predicted}}{\text{Total True Donor Sites in the Test Group}}$$

Specificity:

$$S_p = \frac{\text{Correct True Donor Sites Predicted}}{\text{Total True Donor Sites Predicted}}$$

Thus, Sensitivity is the proportion of actual True Donor Sites in the candidate donor site group that are correctly predicted, and Specificity is the proportion of predicted donor sites that correctly predicted.

2.4.2 Donor Site Detection

At this level, we measured the performance accuracy on a group of DNA sequences for Sensitivity and Specificity. The DNA sequences in this group were those from the Donor Positive and Negative Sequence groups which had been randomly mixed together. To compute sensitivity and specificity, we use the formulas which are similar to those for Donor Site Recognition.

Sensitivity:

$$S_n = \frac{\text{Correct Donor Sites Detected}}{\text{Total Donor Sitess in the Test DNA Group}}$$

Specificity:

$$S_p = \frac{\text{Correct Donor Sites Detected}}{\text{Total Donor Sites Detected}}$$

Thus, Sensitivity is the proportion of actual donor sites in the test DNA sequences that are correctly detected, and Specificity is the proportion of detected donor sites that correctly detected.

CHAPTER 3

DONOR CLASSIFICATION AND DETECTION

3.1 Features of the 5' Splicing Sites

As mentioned in the previous chapters, donor sites are the 5' splice sites for mRNA precursors in higher eucaryotes' genes. The conserved boundary sequences at the 5' splice site (donor site) are shown in Figure 6.

Position:	-3	-2	-1	0	+1	+2	+3	+4	+5
Nucleotide:	C or A	A	G	**G**	**T**	T or A	A	G	T

Figure 6. Consensus sequences for donor site of the splicing junctions in higher eucaryotes. '0' position is the splicing site and it is the first nuclear tide in the intron sequence.

The donor site sequence in Fig.6 just describes the conserved site sequences which is from statistics analysis. Most of the actual donor site sequences differ from it to a greater or lesser degree[25]. Following are the frequencies of the possible four conserved donor site sequences from a group of 550 donor sites:

CAG**GT**TAGT	3 in 550
CAG**GT**AAGT	12 in 550
AAG**GT**TAGT	6 in 550
AAG**GT**AGAT	4 in 550

One can find out the donor site detection is much more complicated than first expected. We also studied the individual nucleotide frequency at each position in the same group of 550 donor sites (see Table 2 and Fig 7).

Table 2. Frequencies of the donor site nucleotides at different positions

Position	A	G	C	T
-3	33.16%	18.02%	36.58%	12.25%
-2	58.20%	16.58%	4.32%	12.07%
-1	8.29%	78.20%	4.32%	9.19%
0	0.00%	100.00%	0.00%	0.00%
1	0.00%	0.00%	0.00%	100.00%
2	48.65%	47.57%	2.16%	1.62%
3	74.05%	11.71%	8.65%	5.59%
4	3.96%	86.13%	4.86%	5.05%
5	17.12%	22.86%	15.14%	44.86%

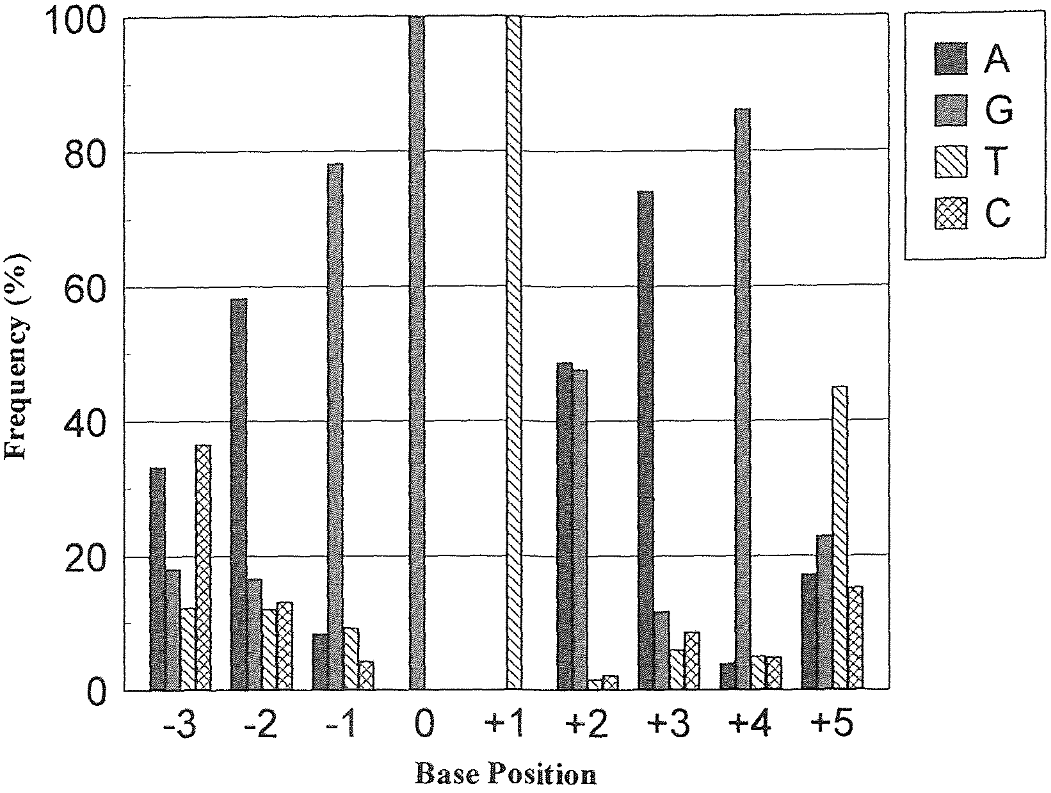

Figure 7. Nucleotide Frequencies in the studied Donor sites at different positions.

From the above frequency data, we can see that the donor site is of degenerate pattern feature. The key point of this Donor detection algorithm is how to mine out the donor degenerate pattern from a set of donor sites containing DNA sequences, and then using pattern match method to classify candidate DNA sequences.

3.2 The Donor Pattern Model

In order to examine donor information, we construct a Donor model as the following.

A donor site contains 10 motifs:

$M_i(p, n)$,

i ($i = 1, 2, \ldots, 10$) denotes motif number (see the following)

p ($1 \leq p \leq 4$) denotes the motif start position in the donor site,

$p = 1$ means the '-3' position in Fig. 1.

n ($6 \leq n \leq 9$) denotes the motif length.

This model says that a donor contains 10 motifs, each motif has a minimum length of 6 nucleotides, and each motifs must contain " GT " nucleotides. Let us look an example.

If a donor site sequence is CAGGTAAGT, we can label it as follows:

Original position:	-1	-2	-3	0	+1	+2	+3	+4	+5
Motif starting position:	1	2	3	4	5	6	7	8	9
Nucleotide:	C	A	G	**G**	**T**	A	A	G	T

According as our Motif model, we can derive 10 motifs as shown in Table 3.

3.2.1 Data Structure for Storage of the Different Motifs

Our program uses the 'set' data structure to hold the same kind of motifs.

Table 3 A sample Motif model

i	$M_i(p, n)$	Sequence	p	n
1	$M_1(1, 9)$	CAG**GT**AAGT	1	9
2	$M_2(1, 8)$	CAG**GT**AAG	1	8
3	$M_3(1, 7)$	CAG**GT**AA	1	7
4	$M_4(1, 6)$	CAG**GT**A	1	6
5	$M_5(2, 8)$	AG**GT**AAGT	2	8
6	$M_6(2, 7)$	AG**GT**AAG	2	7
7	$M_7(2, 6)$	AG**GT**AA	2	6
8	$M_8(3, 7)$	G**GT**AAGT	3	7
9	$M_9(3, 6)$	G**GT**AAG	3	6
10	$M_{10}(4, 6)$	**GT**AAGT	4	6

Note: The bold characters in the sequences are the conserved ‘GT’ dinucleotide.

Let SMp,n represent the set of motifs starting at position 'p' and having length 'n'. *i.e.* $SM_{1,9}$ is the set containing all motifs starting at position '1' and having length '9'.

3.2.2 Donor Motif Library Construction

First, we build up the **Donor Site Learning Group** of 250 **True Donor Sites** as described in Chapter 2. We also call this group **Learning Set**. Then we use the following algorithm to build up the Donor Motif Library (Figure 8).

```
for each donor site in the Learning Set do
    begin
        extract each motifs Mi(p, n), $1 \leq i \leq 10$, $1 \leq p \leq 4$, $6 \leq n \leq 9$;
        insert M(p, n) into the appropriate motif set $SM_{p,n}$
    end;
```

Figure8. Algorithm for building up the Motif Library.

We did an experiment to test if the motif library data can be used for classifying true / false donor sites. First, we construct a group of 200 **True Donor Sites** from another 200 positive donor sequences, using the same method for building up the learning group described above. Then we constructed a group of 900 **False Sites** from 270 of negative donor sequences using the false donor extraction algorithm (Figure 5). The donor site sequence in these two groups are very similar as that ueded in Section 3.3. Next, we extract all the ten motifs from each donor site in these two groups, and searched the Motif

Library for each individual motif to see how many of these motifs from different group could be found in the library. The results were shown in Figure 9. From Figure 9, we can see that the motif library is a very good true donor representative data collection, and it can be used for further study for donor classification.

3.3 Group Discriminant Analysis

For the analysis purpose, we first construct two groups of 9-base DNA sequences data. The first group, **Donor Site Positive Training Group**, contain 250 true donor sites which were extracted from the second 250 positive donor sequences as described Chapter 2. The second group, **Donor Negative Training Group**, was made up of 800 hundred of GT containing non donor sites. All the entries in the negative group were 9 bases long and with GT in the same position as the in the true donor sites (see Table 4).

Next, we introduce a motif score variable, S_i^n, which is defined by motif number (i = 1, 2, ..., 10) and motif length (n = 6, 7, 8, 9).

$$S_i^n = \begin{cases} 2^n & \text{Motif can be found in the Motif Liberary} \\ 0 & \text{Otherwise} \end{cases}$$

Scoring each donor site candidate can be done by calculating the donor score value,

$$S_D = \sum_{i=1}^{10} S_i^n \quad , \quad \text{where} \quad n = 6,\ 7,\ 8,\ 9.$$

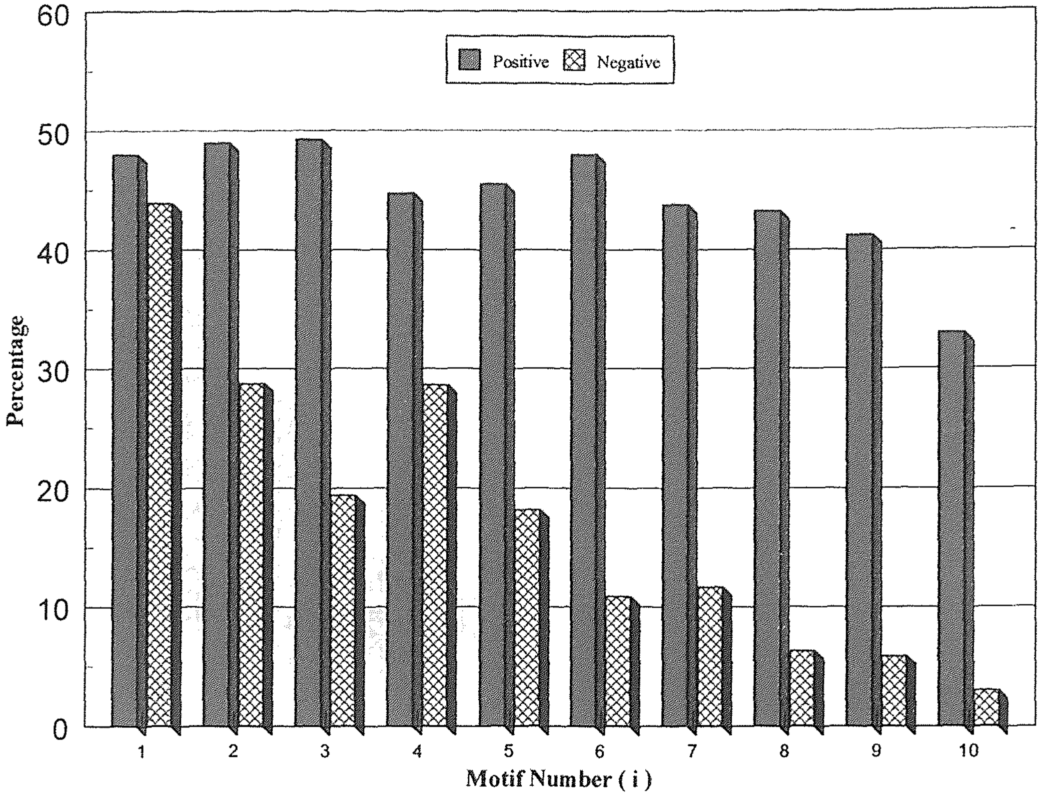

Figure 9. Percentage of motifs from true donor group and false donor group that can be found in the motive library. Positive: motifs from the 200 true donor group. Negative: motifs from the 900 false donor group.

Table 4. The donor site sequences data to be analyzed

No.	Donor sites	Group
1	TAG**GT**GAGA	Positive
2	CCT**GT**AAGT	Positive
3	CAA**GT**AAGG	Positive
4	AAG**GT**ATCA	Positive
.	.	Positive
.	.	Positive
250	AGA**GT**AAGA	Positive
1	TTG**GT**CCAG	Negative
2	GAG**GT**GTCT	Negative
3	ATG**GT**GAAA	Negative
4	CTG**GT**GGAA	Negative
.	.	Negative
.	.	Negative
800	CCA**GT**GAGC	Negative

Finally we use our training algorithm (see Fig. 10) to score all the sequence data in the Positive and Negative donor group, and write the scores into the positive score board or negative score board accordingly. The minimum score in the positive score board is called the positive lower bound, denoted L_p. The maximum score in the negative scoreboard is called negative upper bound, denoted U_n. Let $L_p = \max \{ L_{p,} U_n \}$ and $U_n = \min \{ L_{p,} U_n \}$. For our Positive donor group, we got a maximum score of 1152 and a minimum score of 640 (L_p). For the Negative donor group we got a maximum score of 768 (U_n) and a minimum of score 0. So:

$L_{pos} = \max \{ L_{p,} U_n \} = \max \{ 640, 768 \} = 768;$

$U_{neg} = \min \{ L_{p,} U_n \} = \min \{640, 768 \} = 640.$

3.4 Donor Classification

When classifying candidate donor site sequences, we calculate the score, S_D, based on the motif scores, S_i^n. If $S_D > L_{pos}$ then that candidate donor site is a donor site. If $S_D < U_{neg}$ then it is not a donor site. Otherwise, the " Unknown" verdict is given. Figure 11 shows the donor detecting algorithm.

3.4.1 Donor Classification Performance

The above algorithms were implemented in ANSI C and were tested on a UNIX system and also on a PC system running Window 95. We used the Donor Classify program and classified a group of 692 candidate donor site with 200 known true donor sites among

```
{
 The positive donor training sequences are a group of sequences with
  one donor site in each sequence at the known position P
}

for  each   sequences Seq data in the Positive and Negative groups do
    begin
          extract each motif $M_i(p,n)$;
          search the motif set $SM_{pn}$ for $M_i(p,n)$;
          if  $M_i(p,n)$ found,
              {let S be the score for $M_i(p,n)$}
               S := S + 2^n;
         if the sequence data from the Positive group
               write the Score S to the positive score board;
        else
               write the Score S to the negative score board;
             {let Pmin := the minimum score in the Positive score board;}
             {let Nmax := the maximum score in the Negative score board;}
        if  Pmin < Nmax
             swap (Pmin, Nmax)
     end;
```

Figure 10. Training algorithm

them. None of these true donor sites were extracted from the DNA sequences as the above learning set and positive training group were from. Based on the classification results in Table 5, we calculate the program performance as follows:

Sensitivity for donor site classification:

$$S_n = \frac{\text{True Positive Picked}}{\text{Total Positive}} = \frac{198}{200} = 99.00\%$$

Specificity for donor site classification:

$$S_p = \frac{\text{True Positive Picked}}{\text{Total Positive Picked}} = \frac{198}{216} = 91.66\%$$

3.5 Detection of Donor Site in DNA Sequences

In the real world, when a biologist gets a new DNA sequence, he or she need detect the splicing junction site positions in the DNA. So we modified our Donor site classification algorithm and let the program report the 5' splicing positions (Donor site). The algorithm works like this: scan the input DNA sequence for 'GT' di-nucleotides. If 'GT' found, extract the 9-base candidate donor site. Then input the candidate donor site sequence into the Donor Site Classification program (see Figure 11). If the candidate is a true donor site, report the GT position in the DNA sequence. In order to test the program, we randomly mixed 200 donor positive DNA sequences and 200 donor negative DNA sequences. All these DNA sequences were from that as described in the Chapter 2 but none of them had been used in the above Learning or Training processes. As described in Chapter 2, each of the DNA sequences is 60 base long, and for the 200 donor positive

Table 5. Donor classifying result

Total number of candidate donor sites:	692
Total number of true donor sites:	200
Total true donor sites picked:	216
True Positive (Tp)	198
False Positive (Fp)	18
Truee donor site missed	2
Total false donor sites picked:	476
True Negative (Tn)	474
False Negative (Fn)	2
False donor site missed	18

```
{
  The candidate donor sites group contains one or more 9-base DNA
   sequences with GT dinucleotide at the same position as it in the
   True Donor Site sequences.
}
Input:  candidate donor sites group;
Output: label each candidate donor site sequence
            as True Donor or False Donor or Unknown;
for  each candidate donor site CD do
      begin
         {let S be the score for CD
         S := 0;
          for  motif number $i$ := 1 to $i$ := 10 do
               begin
                    extract motif $M_i(p,n)$;
                    search the motif set $SM_{pn}$ for $M_i(p,n)$;
                    if   $M_i(p,n)$ found,
                           S = S + 2^n;
              end;
          if  S  >=  Pmin
               { label CD as  True Donor; }
          else if  S  <  Nmax
               { label CD as False donor; }
          else
               { label CD as  Unknown; }
      end;
```

Figure 11. Donor Classification algorithm.

DNA sequences, there is only one donor site in each of them. The results are shown in Table 6.

Table 6. Donor detection result.

Total number of input DNA sequences:	400
Total number of donor positive DNA sequences	200
Total number of donor sites in the DNA sequences	200
Total donor sites picked:	204
True Positive (Tp)	186
False Positive (Fp)	18
Positive donor site missed	14

The Sensitivity Sn of this program is:

$$S_n = \frac{\text{True Positive Picked}}{\text{Total Positive}} = \frac{186}{200} = 93.00\%$$

The Specificity Sp of this program is :

$$S_p = \frac{\text{True Positive Picked}}{\text{Total Positive Picked}} = \frac{186}{204} = 91.17\%$$

These results means that the donor detection algorithm can correctly recognize 93% of the total true donor sites at the right splicing positions in tested DNA sequences, and more than 91% of the predicted donor sites are correct.

CHAPTER 4

DISCUSSION

Computational gene identification will play an increasingly important role as the human genome project enters the large-scale sequencing phase[8], and a number of computational methods have recently been developed. As Moises Burset and Roderic Guigo indicated that programs currently available are far from being powerful enough to elucidate genomic structure completely[7]. In order to develop more accurate and efficient programs for detecting gene structures, in this thesis, we concentrate on splicing junction donor classification and detection. Using our motif model we can discover the degenerate pattern features of the splicing junction sites to a great degree. Based on this model, our donor detection algorithm can correctly recognize 93% of the total donor site in the test group. And more than 91% of the donor sites picked by our program are correct. These precision rates are higher than the best existing donor classification algorithm[24]. This research made a very important progress toward our full gene structure detection algorithm development.

The pattern recognition theory are based on the assumption that in the genome there is a tendency to avoid nonfunctionıng signal-like sites. Thus, researchers usually using a learning sample consisting of two classes, sites and nonsites, the later class usually formed by random fragments of the natural DNA. For example, Yoichi Iida extracted all the possible 9-base sequences other than the 5'-splice sites from rabbit β-globin pre-mRNA as his non-donor site group for his quantification analysis of 5'-splice signal[6]. In contrast to that approach, our non-splice junction sites groups did not formed by random fragments of the natural DNA. All the entries in our non-donor site sequence group (we call this group False Donor Site Group) contained the dinucleotide GT at the same position as in the real donor sites. Because the GT dinucleotide is almost invariable in all donor sites, we treat only GT dinucleotide containing fragments in the genome DNA as candidate splicing junction sites. In this way, our approach works well to discriminate between the true splicing sites and non-splicing sites.

Most of the current programs available for gene structure detection do not work well for short DNA sequences. For example, GRAIL2 can not identify the exon boundaries in a DNA sequence shorter than 100 bases. This is not a limitation for our method, because our motif model is based on the sequence information in the splicing junctions. The splicing junction detection algorithms did not consider global sequence information. People may argue about this. But we think, because all the exons have acceptor, donor or both sites, if we can pinpoint precisely the splicing junctions, we will get good results for detection exons from the genomic DNA sequence. Of cause, we will integrate other sequence information such as coding potential into our full gene detection program, but we expect the splicing junction detection algorithms will be the backbone of our applications.

Mosises Burset and Roderic Guigo also studied the relative robustness of gene structure prediction programs to sequencing errors. Indeed, artifactual nucleotide insertions and deletions do occur while sequencing DNA. Moreover, plans are underway to speed up the full-length sequencing of the human genome by allowing lower sequence accuracy (8). In Mosises Burset and Roderic Guigo's results, for most of the programs they evaluated, the accuracy decreased almost 50% when the mutated DNA sequences were presented to those programs for exon detection. Like GRAIL2, accuracy for the original DNA sequences was 40% for exon detection, but it just could recognize 21% of the exons in the same group of DNA sequences with 1% of mutations[7]. We think this is because those programs more depended on the coding potential in the DNA sequences. The mutation caused frame shifts and affected program performances to a great degree. We expected this will also not a big problem for our method, because we have concentrated on the information from the splicing junction sites. If the mutation does not occur inside the splice sites, it will not affect the performance much. If this is true, it will be of real value when used in the large-scale sequencing projects.

As mentioned above, this research made a very good step toward our full gene structure detection program development. This is not only because the donor classification / detection results are very promising, our motif model approach and degenerate pattern match algorithms for the donor classification will also fit our further research. Based on this study, it may be easy to build up a acceptor motif model for the classification and detection of the 3' splicing junction sites in the eukaryotic genes. Certainly, we will also try these methods for the detection of translation initiation signal sequence of mRNA (start coden signal). So we think the method developed in this research is the benchmark for our future studies.

APPENDIX A

LEARNING AND TRAINING PROGRAM FOR DONOR CLASSIFICATION

This is the ANSI C version of the program for the donor classification Learning and Training. Algorithms that were implemented into this program include Algorithm for True Donor Extraction, Algorithm for False Donor Extraction, Learning Algorithm, Training Algorithms etc.

```
/* Program Name: DonorLearnAndTrain.c*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAXLENGTH 100
#define MaxSeq 500
#define MaxNum 300

#define point6 64
#define point7 128
#define point8 256
#define point9 512
typedef struct _donor
        {
                char    b1_6[7], b1_7[8], b1_8[9], b1_9[10],
                        b2_6[7], b2_7[8], b2_8[9],
                        b3_6[7], b3_7[8],
                        b4_6[7];
        } donor;

typedef struct _motif
        {
                int occur;
                int posoccur;
                int negoccur;
                char seq[10];
        } motif;

typedef struct _reccord
       {
                int total;   /* Total num of entries*/
                motif list[MaxNum];
        } reccord;

donor Donor;
int TotalSeq;

int  Score;

int ScoreBoard[MaxSeq];
int negScoreBoard[MaxSeq];

int  pmin, pmax,  /*min and max scores for positive donor*/
     nmin, nmax;  /*min and  max scores for nagative donor*/


reccord
        motif_b1_6,
        motif_b1_7,
        motif_b1_8,
        motif_b1_9,
        motif_b2_6,
        motif_b2_7,
        motif_b2_8,
        motif_b3_6,
        motif_b3_7,
        motif_b4_6;


FILE *F_In, *F_Out;

main()
{

        char *donor_bas = "donobas.dat";
        char *donor_dat = "donodat.dat";
        char *Score_out = "scoreout.dat";
        char *donor_pos = "donopos.dat";
        char *donor_neg = "dononeg.dat";

        FILE *F_pos, *F_neg, *F_Out;

        initiateData();

        if ((F_In = fopen(donor_bas, "r")) == NULL)
        {
```

```
        printf("ERROR: %s file open.\n", donor_bas);
        exit (0);
}

/*Learning */
LearnDonor();

/* Tarining */
if ((F_pos = fopen (donor_pos, "r")) == NULL)
{
        printf("ERROR: Open file %s. \n", donor_pos);
        exit (0);
}

if ((F_Out = fopen (Score_out, "w")) == NULL)
{
        printf("ERROR: Open file %s. \n", Score_out);
        exit (0);
}

/*Tain the postive donor sequences.*/
TrainPosDonor(F_pos, ScoreBoard);

pmin = PosiveLowBound(ScoreBoard, TotalSeq, &pmax);


WriteScoreToFile(ScoreBoard, TotalSeq, F_Out);
if ((F_neg = fopen(donor_neg, "r")) == NULL)
{
        printf("ERROR: Open file %s. \n", donor_neg);
        exit (0);
}

/* Tarin the negative sequences*/
TrainNegDonor(F_neg, negScoreBoard);
nmax = NegativeHighBound(negScoreBoard, TotalSeq, &nmin);
WriteScoreToFile(negScoreBoard, TotalSeq, F_Out);

WriteDonorToFile(donor_dat);

printf("\nThe Positive Low Bound is: %ld.\n", pmin);
printf("\nThe Positive High Bound is: %ld.\n", pmax);

printf("\nThe Negative High Bound is: %ld.\n", nmax);
printf("\nThe Negative Low Bound is: %ld.\n", nmin);
```

```
        fclose(F_In);
        fclose(F_pos);
        fclose(F_neg);
        fclose(F_Out);



        return 0;
}



void initiateData()
{
        Donor.b1_6[7] = '\0';
        Donor.b1_7[8]= '\0';
        Donor.b1_8[9]= '\0';
        Donor.b1_9[10]= '\0';
        Donor.b2_6[7]= '\0';
        Donor.b2_7[8]= '\0';
        Donor.b2_8[9]= '\0';
        Donor.b3_6[7]= '\0';
        Donor.b3_7[8]= '\0';
        Donor.b4_6[7]= '\0';

        motif_b1_6.total = 0;
        motif_b1_7.total = 0;
        motif_b1_8.total = 0;
        motif_b1_9.total = 0;
        motif_b2_6.total = 0;
        motif_b2_7.total = 0;
        motif_b2_8.total = 0;
        motif_b3_6.total = 0;
        motif_b3_7.total = 0;
        motif_b4_6.total = 0;


}
/* ReccordDonor() write the subdonor sequences into the reccord list.
        If a subsequence is already there, just increase the occrence.
        Else if the subsequence is not in the list yet, write the
        subsequence into the list and initiate its occurence to 1.
*/
void ReccordDonor(reccord *R, char *S)
```

```
{
        int i;
        int found;
        if (R->total == 0)
        {
                strcpy(R->list[0].seq, S);
                R->list[0].occur = 1;
                R->list[0].negoccur=0;
                R->list[0].posoccur=0;
                R->total = 1;
        }

        else
        {
                found = 0;
                for (i = 0; i < R->total; i++ )
                {
                        if (strcmp(R->list[i].seq, S) == 0)
                        {
                                R->list[i].occur++;
                                R->list[i].negoccur=0;
                                R->list[i].posoccur=0;
                                found = 1;
                                break;
                        }
                }
                if (found == 0)
                {
                        strcpy(R->list[R->total].seq, S);
                        R->list[R->total].occur = 1;
                        R->list[R->total].negoccur=0;
                        R->list[R->total].posoccur=0;
                        R->total++;
                }
        } /*else*/
}  /* ReccordDonor*/

void ExtractMotif(int p,  char A[], donor *motif)
{
        int first, last;
        int i, m;
        last = p + 5;
        first = p - 3;

        for (i = first; i <= last; i++)
```

```
{
        m = i - first;

        switch (m)
        {
                case 0:
                        motif->b1_9[0] = motif->b1_8[0]
                        =motif->b1_7[0] = motif->b1_6[0] = A[i];
                        break;
                case 1:
                        motif->b1_9[1] = motif->b1_8[1]
                        =motif->b1_7[1] = motif->b1_6[1]
                        =motif->b2_8[0] = motif->b2_7[0]
                        = motif->b2_6[0] = A[i];
                        break;
                case 2:
                        motif->b1_9[2] = motif->b1_8[2]
                        =motif->b1_7[2] = motif->b1_6[2]
                        =motif->b2_8[1] = motif->b2_7[1]
                        =motif->b2_6[1]
                        =motif->b3_6[0] = motif->b3_7[0] =  A[i];
                        break;
                case 3:
                        motif->b1_9[3] = motif->b1_8[3]
                        =motif->b1_7[3] = motif->b1_6[3]
                        =motif->b2_8[2] = motif->b2_7[2]
                        =motif->b2_6[2]
                        =motif->b3_6[1] = motif->b3_7[1]
                      =motig->b4_6[0] = A[i] ;
                        break;
                case 4:
                        motif->b1_9[4] = motif->b1_8[4]
                        =motif->b1_7[4] = motif->b1_6[4]
                        =motif->b2_8[3] = motif->b2_7[3]
                        =motif->b2_6[3]
                        =motif->b3_6[2] = motif->b3_7[2]
                      =motig->b4_6[1] = A[i];
                        break;
                case 5:
                        motif->b1_9[5] = motif->b1_8[5]
                        =motif->b1_7[5] = motif->b1_6[5]
                        =motif->b2_8[4] = motif->b2_7[4]
                        =motif->b2_6[4]
                        =motif->b3_6[3] = motif->b3_7[3]
                      =motig->b4_6[2] = A[i] ;
```

```
                                        break;
                                case 6:
                                        motif->b1_9[6] = motif->b1_8[6]
                                        =motif->b1_7[6]
                                        =motif->b2_8[5] = motif->b2_7[5]
                                        =motif->b2_6[5]
                                        =motif->b3_6[4] = motif->b3_7[4]
                                       =motig->b4_6[3] = A[i] ;
                                        break;
                                 case 7:
                                        motif->b1_9[7] = motif->b1_8[7]
                                        =motif->b2_8[6] = motif->b2_7[6]
                                        =motif->b3_6[5] = motif->b3_7[5]
                                       =motig->b4_6[4] = A[i] ;
                                        break;
                                 case 8:
                                        motif->b1_9[8]
                                        =motif->b2_8[7]
                                        = motif->b3_7[6]
                                        =motif->b4_6[5] =  A[i];
                                        break;
                        }  /*switch*/
                }
        }


        void WriteMotifToList( donor *motif)
        {
                /*write motif to the reccords*/
                ReccordDonor(&motif_b1_6, motif->b1_6);
                ReccordDonor(&motif_b2_6, motif->b2_6);
                ReccordDonor(&motif_b3_6, motif->b3_6);
                ReccordDonor(&motif_b4_6, motif->b4_6);
                ReccordDonor(&motif_b1_7, motif->b1_7);
                ReccordDonor(&motif_b2_7, motif->b2_7);
                ReccordDonor(&motif_b3_7, motif->b3_7);
                ReccordDonor(&motif_b1_8, motif->b1_8);
                ReccordDonor(&motif_b2_8, motif->b2_8);
                ReccordDonor(&motif_b1_9, motif->b1_9);
        }



        /*Input sequences from a file*/
        void LearnDonor()
```

```
{
      int i, j , n,
      seqlen,   /*length of input seq*/
      TotalSeq;  /*Number of sequences in th input file*/

       char seq[MAXLENGTH],
            T[MAXLENGTH],
            ch;
       for (TotalSeq =0; ;)
       {
             if ((fgets(T, MAXLENGTH, F_In)) == NULL)
                    break;
             if (T[0] == '>')
                    TotalSeq++;
       }
       /****/
       printf("\nTotalSeq: %i\n", TotalSeq);
      if (TotalSeq > MaxSeq)
      {
             printf("\n%s\n", "Error: Too many sequences.");
             exit(0);
      }
      rewind(F_In);
      for (n = 0; n < TotalSeq; n++)
      {
             if (fgets(T, MAXLENGTH, F_In) != NULL)
             {
                    if (T[0] == '>')
                    {
                           n--;  /*do not count sequence name line*/
                           continue;
                    }
                    i = 0;
                    seqlen = strlen(T);
                    if (T[seqlen -1] == '\n')
                           seqlen--;
                    for (j = 0;  j < seqlen; j ++)
                    {
                           ch = T[j];
                           if ( (ch == ' ') || (ch == '*')||(ch == '\t'))
                                  continue;
                           if ((ch >= 'a') && (ch <= 'z'))
                                  ch = ch + 'A' - 'a' ;  /* Change into capital*/
                           if ((ch < 'A') || (ch > 'Z'))
                           {
```

```
                    printf("Err input!, %i\n", n);
                /*  exit(0);     */
                }
                seq[i++]  = ch;
                if (i>(MAXLENGTH -2))
                {
                    printf("Err: Sequence is too long.\n");
                    exit(0);
                }

            }


            ExtractMotif(30, seq, &Donor);
            WriteMotifToList( &Donor);

        }
    }
}




/*Write the donor data to a file*/
void WriteDonorToFile(char *fileName)
{
    int i;

    if ((F_Out = fopen(fileName, "w")) == NULL)
    {
        printf("\nError: open %s file", fileName);
        exit(0);
    }


    fprintf(F_Out, "\n\n**motif_b1_6**\n");
    fprintf(F_Out, "Total:  %i\n", motif_b1_6.total);
    for( i = 0; i < motif_b1_6.total; i++ )
        fprintf(F_Out, "\t%s\t%i\t%i\t%i\n",
            motif_b1_6.list[i].seq, motif_b1_6.list[i].occur,
            motif_b1_6.list[i].posoccur,
            motif_b1_6.list[i].negoccur);

    fprintf(F_Out, "\n\n**motif_b2_6**\n");
    fprintf(F_Out, "Total:  %i\n", motif_b2_6.total);
```

```
for( i = 0; i < motif_b2_6.total; i++ )
        fprintf(F_Out, "\t%s\t%i\t%i\t%i\n",
                motif_b2_6.list[i].seq, motif_b2_6.list[i].occur,
                motif_b2_6.list[i].posoccur,
                motif_b2_6.list[i].negoccur);

fprintf(F_Out, "\n\n**motif_b3_6**\n");
fprintf(F_Out, "Total:  %i\n", motif_b3_6.total);
for( i = 0; i < motif_b3_6.total; i++ )
        fprintf(F_Out, "\t%s\t%i\t%i\t%i\n",
                motif_b3_6.list[i].seq, motif_b3_6.list[i].occur,
                motif_b3_6.list[i].posoccur,
                motif_b3_6.list[i].negoccur);

fprintf(F_Out, "\n\n**motif_b4_6**\n");
fprintf(F_Out, "Total:  %i\n", motif_b4_6.total);
for( i = 0; i < motif_b4_6.total; i++ )
        fprintf(F_Out, "\t%s\t%i\t%i\t%i\n",
                motif_b4_6.list[i].seq, motif_b4_6.list[i].occur,
                motif_b4_6.list[i].posoccur,
                motif_b4_6.list[i].negoccur);

fprintf(F_Out, "\n\n**motif_b1_7**\n");
fprintf(F_Out, "Total:  %i\n", motif_b1_7.total);
for( i = 0; i < motif_b1_7.total; i++ )
        fprintf(F_Out, "\t%s\t%i\t%i\t%i\n",
                motif_b1_7.list[i].seq, motif_b1_7.list[i].occur,
                motif_b1_7.list[i].posoccur,
                motif_b1_7.list[i].negoccur);

fprintf(F_Out, "\n\n**motif_b2_7**\n");
fprintf(F_Out, "Total:  %i\n", motif_b2_7.total);
for( i = 0; i < motif_b2_7.total; i++ )
        fprintf(F_Out, "\t%s\t%i\t%i\t%i\n",
                motif_b2_7.list[i].seq, motif_b2_7.list[i].occur,
                motif_b2_7.list[i].posoccur,
                motif_b2_7.list[i].negoccur);

fprintf(F_Out, "\n\n**motif_b3_7**\n");
fprintf(F_Out, "Total:  %i\n", motif_b3_7.total);
for( i = 0; i < motif_b3_7.total; i++ )
        fprintf(F_Out, "\t%s\t%i\t%i\t%i\n",
                motif_b3_7.list[i].seq, motif_b3_7.list[i].occur,
                motif_b3_7.list[i].posoccur,
                motif_b3_7.list[i].negoccur);
        fprintf(F_Out, "\n\n**motif_b1_8**\n");
        fprintf(F_Out, "Total:  %i\n", motif_b1_8.total);
        for( i = 0; i < motif_b1_8.total; i++ )
                fprintf(F_Out, "\t%s\t%i\t%i\t%i\n",
                        motif_b1_8.list[i].seq, motif_b1_8.list[i].occur,
                        motif_b1_8.list[i].posoccur,
                        motif_b1_8.list[i].negoccur);

        fprintf(F_Out, "\n\n**motif_b2_8**\n");
        fprintf(F_Out, "Total:  %i\n", motif_b2_8.total);
        for( i = 0; i < motif_b2_8.total; i++ )
                fprintf(F_Out, "\t%s\t%i\t%i\t%i\n",
                        motif_b2_8.list[i].seq, motif_b2_8.list[i].occur,
                        motif_b2_8.list[i].posoccur,
                        motif_b2_8.list[i].negoccur);


        fprintf(F_Out, "\n\n**motif_b1_9**\n");
        fprintf(F_Out, "Total:  %i\n", motif_b1_9.total);
        for( i = 0; i < motif_b1_9.total; i++ )
                fprintf(F_Out, "\t%s\t%i\t%i\t%i\n",
                        motif_b1_9.list[i].seq, motif_b1_9.list[i].occur,
                        motif_b1_9.list[i].posoccur,
                        motif_b1_9.list[i].negoccur);
}

/************ Training Portion Begin****************/

void InitiateScoreBoard()
{
        int i;
        for (i = 0; i < MaxSeq; i++ )
        {
                ScoreBoard[i] = 0;
                negScoreBoard[i] = 0;
        }
}

void VoteDonorPos(donor *D)
{
        int i;
        Score = 0;
```

```
  for (i = 0; i < motif_b1_6.total; i++ )
  {
          if (strcmp(motif_b1_6.list[i].seq, D->b1_6) == 0)
          {
                  Score += point6;/*motif_b6.list[i].occur;*/
                  motif_b1_6.list[i].posoccur++;
          }
  }

  for (i = 0; i < motif_b2_6.total; i++ )
  {
          if (strcmp(motif_b2_6.list[i].seq, D->b2_6) == 0)
          {
                  Score += point6;/*motif_b6.list[i].occur;*/
                  motif_b2_6.list[i].posoccur++;
          }
  }

  for (i = 0; i < motif_b3_6.total; i++ )
  {
          if (strcmp(motif_b3_6.list[i].seq, D->b3_6) == 0)
          {
                  Score += point6;/*motif_b6.list[i].occur;*/
                  motif_b3_6.list[i].posoccur++;
          }
 }

for (i = 0; i < motif_b4_6.total; i++ )
{
          if (strcmp(motif_b4_6.list[i].seq, D->b4_6) == 0)
          {
                  Score += point6;/*motif_b6.list[i].occur;*/
                  motif_b4_6.list[i].posoccur++;
          }
  }

  for (i = 0; i < motif_b1_7.total; i++ )
  {
          if (strcmp(motif_b1_7.list[i].seq, D->b1_7) == 0)
          {
                  Score += point7;/*motif_b7.list[i].occur;*/
                  motif_b1_7.list[i].posoccur++;
          }
  }
 for (i = 0; i < motif_b2_7.total; i++ )
 {

          if (strcmp(motif_b2_7.list[i].seq, D->b2_7) == 0)
          {
                   Score += point7;/*motif_b7.list[i].occur;*/
                   motif_b2_7.list[i].posoccur++;
          }
 }

 for (i = 0; i < motif_b3_7.total; i++ )
{
          if (strcmp(motif_b3_7.list[i].seq, D->b3_7) == 0)
          {
                   Score += point7;/*motif_b7.list[i].occur;*/
                   motif_b3_7.list[i].posoccur++;
          }
 }


 for (i = 0; i < motif_b1_8.total; i++ )
 {
          if (strcmp(motif_b1_8.list[i].seq, D->b1_8) == 0)
          {
                   Score += point8;/*motif_b8.list[i].occur;*/
                   motif_b1_8.list[i].posoccur++;
          }
 }

 for (i = 0; i < motif_b2_8.total; i++ )
 {
          if (strcmp(motif_b2_8.list[i].seq, D->b2_8) == 0)
          {
                   Score += point8;/*motif_b8.list[i].occur;*/
                   motif_b2_8.list[i].posoccur++;
          }
 }

 for (i = 0; i < motif_b1_9.total; i++ )
 {
          if (strcmp(motif_b1_9.list[i].seq, D->b1_9) == 0)
          {
                   Score += point9;/*motif_b9.list[i].occur;*/
                   motif_b1_9.list[i].posoccur++;
                   break;
```

```
        }
    }
}

void VoteDonorNeg(donor *D)
{
    int i;
    Score = 0;

    for (i = 0; i < motif_b1_6.total; i++ )
    {
        if (strcmp(motif_b1_6.list[i].seq, D->b1_6) == 0)
        {
            Score += point6;/*motif_b6.list[i].occur;*/
            motif_b1_6.list[i].negoccur++;
        }
    }

    for (i = 0; i < motif_b2_6.total; i++ )
    {
        if (strcmp(motif_b2_6.list[i].seq, D->b2_6) == 0)
        {
            Score += point6;/*motif_b6.list[i].occur;*/
            motif_b2_6.list[i].negoccur++;
        }
    }

    for (i = 0; i < motif_b3_6.total; i++ )
    {
        if (strcmp(motif_b3_6.list[i].seq, D->b3_6) == 0)
        {
            Score += point6;/*motif_b6.list[i].occur;*/
            motif_b3_6.list[i].negoccur++;
        }
   }

   for (i = 0; i < motif_b4_6.total; i++ )
   {
        if (strcmp(motif_b4_6.list[i].seq, D->b4_6) == 0)
        {
            Score += point6;/*motif_b6.list[i].occur;*/
            motif_b4_6.list[i].negoccur++;
        }
   }
```

```
for (i = 0; i < motif_b1_7.total; i++ )
{
          if (strcmp(motif_b1_7.list[i].seq, D->b1_7) == 0)
          {
                  Score += point7;/*motif_b7.list[i].occur;*/
                  motif_b1_7.list[i].negoccur++;
          }
 }

  for (i = 0; i < motif_b2_7.total; i++ )
  {
          if (strcmp(motif_b2_7.list[i].seq, D->b2_7) == 0)
          {
                  Score += point7;/*motif_b7.list[i].occur;*/
                  motif_b2_7.list[i].negoccur++;
          }
  }

  for (i = 0; i < motif_b3_7.total; i++ )
 {
          if (strcmp(motif_b3_7.list[i].seq, D->b3_7) == 0)
          {
                  Score += point7;/*motif_b7.list[i].occur;*/
                  motif_b3_7.list[i].negoccur++;
          }
  }


  for (i = 0; i < motif_b1_8.total; i++ )
  {
          if (strcmp(motif_b1_8.list[i].seq, D->b1_8) == 0)
          {
                  Score += point8;/*motif_b8.list[i].occur;*/
                  motif_b1_8.list[i].negoccur++;
          }
  }

  for (i = 0; i < motif_b2_8.total; i++ )
  {
          if (strcmp(motif_b2_8.list[i].seq, D->b2_8) == 0)
          {
                  Score += point8;/*motif_b8.list[i].occur;*/
                  motif_b2_8.list[i].negoccur++;
          }
  }
```

```
        for (i = 0; i < motif_b1_9.total; i++ )
        {
                if (strcmp(motif_b1_9.list[i].seq, D->b1_9) == 0)
                {
                        Score += point9;/*motif_b9.list[i].occur;*/
                        motif_b1_9.list[i].negoccur++;
                        break;
                }
        }
  }


/*Input sequences from a file*/
void TrainNegDonor(FILE *F_In, int ScoreBoard[])
{
        int i,n,p,
       seqlen; /*length of input seq*/

        int TempScor;

         char seq[MAXLENGTH],
                T[MAXLENGTH];

         for (TotalSeq =0; ;)
         {
                if ((fgets(T, MAXLENGTH, F_In)) == NULL)
                        break;
                if (T[0] == '>')
                        TotalSeq++;
        }

         printf("\nTotalSeq: %i\n", TotalSeq);
        if (TotalSeq > MaxSeq)
        {
                printf("\n%s\n", "Error: Too many sequences.");
                exit(0);
        }
        rewind(F_In);
        for (n = 0; n < TotalSeq; n++)
        {
                if (fgets(T, MAXLENGTH, F_In) != NULL)
                {
                        if (T[0] == '>')
                        {
```

```
                n--;  /*do not count sequence name line*/
                continue;
            }
            i = 0;
            seqlen = strlen(T);
            if (T[seqlen -1] == '\n')
                seqlen--;

           strcpy(seq, T);
            seqlen = strlen(seq);
            TempScor = 0;

            for (i=0; i < seqlen; i++)
            {

                if (i < 3)
                    continue;
                if ((seq[i] == 'G') && (seq[i+1] == 'T'))
                {
                   Score = 0;
                    p = i;
                    ExtractMotif(p, seq, &Donor);
                    VoteDonorNeg(&Donor);

                    TempScor = Score;
                    if (TempScor > ScoreBoard[n])
                        ScoreBoard[n] = TempScor;
                }
            }
        }
    }
}

/*Input sequences from a file*/
void TrainPosDonor(FILE *F_In, int ScoreBoard[])
{
    int i,n,
    seqlen;   /*length of input seq*/

    int TempScor;

     char seq[MAXLENGTH],
                T[MAXLENGTH];

     for (TotalSeq =0; ;)
    {
        if ((fgets(T, MAXLENGTH, F_In)) == NULL)
            break;
        if (T[0] == '>')
            TotalSeq++;
    }

    printf("\nTotalSeq: %i\n", TotalSeq);
    if (TotalSeq > MaxSeq)
    {
        printf("\n%s\n", "Error: Too many sequences.");
        exit(0);
    }
    rewind(F_In);
    for (n = 0; n < TotalSeq; n++)
    {
        if (fgets(T, MAXLENGTH, F_In) != NULL)
        {
            if (T[0] == '>')
            {
                n--;  /*do not count sequence name line*/
                continue;
            }
            i = 0;
            seqlen = strlen(T);
            if (T[seqlen -1] == '\n')
                seqlen--;

            strcpy(seq, T);
            seqlen = strlen(seq);
            TempScor = 0;
            Score = 0;

            ExtractMotif(30 , seq, &Donor);

            VoteDonorPos( &Donor);

            TempScor = Score;
            if (TempScor > ScoreBoard[n])
                ScoreBoard[n] = TempScor;
        }
    }
}
```

```
void WriteScoreToFile(int A[], int Total, FILE *F_Out)
{
        int i;
        for (i = 0; i < Total; i++)
        {
                fprintf(F_Out, "\t%i \t %ld\n", i+1, A[i]);
        }
}

int PosiveLowBound(int A[], int T, int *pmax)
{
        int min;
        int i;
        min = A[0];
        *pmax = A[0];
        for (i = 0; i < T; i++)
        {
                if (min > A[i])
                        min = A[i];
                if (*pmax < A[i])
                        *pmax = A[i];
        }

        return min;
}

int NegativeHighBound(int A[], int T,int *nmin)
{
        int max;
        int i;
        max = *nmin = A[0];
        for (i = 0; i < T; i++ )
        {
                if (max < A[i])
                      max = A[i];
                if (*nmin > A[i])
                     *nmin = A[i];
        }

        return max;
}
```

APPENDIX B

DONOR CLASSIFICATION PROGRAM

This is the ANSI C version of the Donor Classification program
The defined MAXLENGTH and MaxSeq can be changed into different values. But other defined constant values can not be changed.

```
/* Program Name: DonorClassify.c */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAXLENGTH 1000
#define MaxSeq 300

#define point6 64
#define point7 128
#define point8 256
#define point9 512
typedef struct _donor
            {
                    char b1_6[7], b1_7[8], b1_8[9],
                    b2_6[7], b2_7[8], b2_8[9],
                     b3_6[7], b3_7[8],
                     b4_6[7];
            } donor;

donor Donor;
int TotalSeq;

typedef struct grade
          {
                    int Score;
                    char *Class;
          } Grade;
int Score;

Grade ScoreBoard[MaxSeq];

#define Pos_Min 768      /*Positive lower bound, inclusive*/
```

```
#define Neg_Max 640     /*Negative high bound*/
#define Positive "Donor"
#define Negative "Non-Donor"
#define Unknown "Unknown"

#define numb1_6 68
#define numb1_7 141
#define numb1_8 183
#define numb2_6 49
#define numb2_7 90
#define numb2_8 171
#define numb3_6 40
#define numb3_7 102
#define numb4_6 69
char
        motif_b1_6[numb1_6][7],
        motif_b1_7[numb1_7][8],
        motif_b1_7[numb1_7][8],
        motif_b1_7[numb1_7][8],
        motif_b1_8[numb1_8][9],
        motif_b2_6[numb2_6][7],
        motif_b2_7[numb2_7][8],
        motif_b2_8[numb2_8][9],
        motif_b3_6[numb3_6][7],
        motif_b3_7[numb3_7][8],
        motif_b4_6[numb4_6][7];

FILE  *Lib_In, *F_In, *F_Out, *Score_Out;

main()
{
        char sequence[50];
        char *donor_lib = "donordat.lib";
        char Score_file[50];
        char *Dono_out = "dono_out2.doc";
        printf("\n\n\t\tWelcome for using this\n");
        printf("\t\tDonor Classify program!\n");
        printf("\n\t\tNote:");
        printf("\n\t1) Please put the sequence file in the\n");
        printf("\t   same directory as this program is in.\n");
        printf("\t2) The max number of sequences is %i\n", MaxSeq);
        printf("\t3) The results will be in the file name you specifed\n");

        initiateData();
        if ((Lib_In = fopen(donor_lib, "r")) == NULL)
```

```
	{
		printf("ERROR: %s file open.\n", donor_lib);
		exit (0);
	}

	InportDonorLib();
	fclose(Lib_In);

	printf("\nPlease enter your sequence file:\n");
	scanf("%49s", sequence);

	if ((F_In = fopen(sequence, "r")) == NULL)
	{
		printf("ERROR: %s file open.\n", sequence);
		exit (0);
	}

	Classif(F_In, ScoreBoard);
	fclose(F_In);

	printf("\nPlease enter your output file  to hold the results:\n");

	scanf("%49s", Score_file);

	if ((Score_Out = fopen(Score_file, "w")) == NULL)
	{
		printf("ERROR: %s file open.\n", Score_file);
		exit (0);
	}
	WriteScoreToFile(ScoreBoard, TotalSeq, Score_Out);
	fclose(Score_Out);

	printf("\nThe program finished for Donor Classification.\n");
	printf("Please look at your results in file: %s.\n",Score_file);

	return 0;
}


void initiateData()
{
	Donor.b1_6[6] = '\0';
	Donor.b1_7[7]= '\0';
	Donor.b1_8[8]= '\0';
	Donor.b2_6[6]= '\0';
```

```
        Donor.b2_7[7]= '\0';
        Donor.b2_8[8]= '\0';
        Donor.b3_6[6]= '\0';
        Donor.b3_7[7]= '\0';
        Donor.b4_6[6]= '\0';

}

void InportDonorLib()
{
        int i, j;
        char T[15],T1[5];
        for (i = 0; i < numb1_6; i++)
        {
                if (fgets(T, 10, Lib_In) == NULL)

                {
                        printf("Error1: Donor Lib file open.\n");
                        exit(0);
                }
                strcpy(T1, T);
                for(j = 0; j < 6; j++)
                        motif_b1_6[i][j] = T1[j];
                motif_b1_6[i][6]='\0';

        }

        for (i = 0; i < numb2_6; i++)
        {
                if (fgets(T, 10, Lib_In) == NULL)
                {
                        printf("Error2: Donor Lib file open.\n");
                        exit(0);
                }
                strcpy(T1, T);
                for(j = 0; j < 6; j++)
                        motif_b2_6[i][j] = T1[j];
                motif_b2_6[i][6]='\0';

        }

        for (i = 0; i < numb3_6; i++)
        {
                if (fgets(T, 11, Lib_In) == NULL)
                {
```

```
                printf("Error3: Donor Lib file open.\n");
                exit(0);
        }
        strcpy(T1, T);
        for(j = 0; j < 6; j++)
                motif_b3_6[i][j] = T1[j];
        motif_b3_6[i][6]='\0';

}

for (i = 0; i < numb4_6; i++)
{
        if (fgets(T, 10, Lib_In) == NULL)
        {
                printf("Error: Donor Lib file open.\n");
                exit(0);
        }
        strcpy(T1, T);
        for(j = 0; j < 6; j++)
                motif_b4_6[i][j] = T1[j];
        motif_b4_6[i][6]='\0';
}


for (i = 0; i < numb1_7; i++)
{
        if (fgets(T, 10, Lib_In) == NULL)
        {
                printf("Error: Donor Lib file open.\n");
                exit(0);
        }
        strcpy(T1, T);
        for(j = 0; j < 7; j++)
                motif_b1_7[i][j] = T1[j];
        motif_b1_7[i][7]='\0';
}

for (i = 0; i < numb2_7; i++)
{
        if (fgets(T, 10, Lib_In) == NULL)
        {
                printf("Error: Donor Lib file open.\n");
                exit(0);
        }
        strcpy(T1, T);
        for(j = 0; j < 7; j++)
            motif_b2_7[i][j] = T1[j];
        motif_b2_7[i][7]='\0';
    }

    for (i = 0; i < numb3_7; i++)
    {
        if (fgets(T, 10, Lib_In) == NULL)
        {
            printf("Error: Donor Lib file open.\n");
            exit(0);
        }
        strcpy(T1, T);
        for(j = 0; j < 7; j++)
            motif_b3_7[i][j] = T1[j];
        motif_b3_7[i][7]='\0';
    }


    for (i = 0; i < numb1_8; i++)
    {
        if (fgets(T, 12, Lib_In) == NULL)
        {
            printf("Error: Donor Lib file open.\n");
            exit(0);
        }
        strcpy(T1, T);
        for(j = 0; j < 8; j++)
         motif_b1_8[i][j] = T1[j];
        motif_b1_8[i][8]='\0';
    }

    for (i = 0; i < numb2_8; i++)
    {
        if (fgets(T, 12, Lib_In) == NULL)
        {
            printf("Error: Donor Lib file open.\n");
            exit(0);
        }
        strcpy(T1, T);
        for(j = 0; j < 8; j++)
            motif_b2_8[i][j] = T1[j];
        motif_b2_8[i][8]='\0';
    }
}
```

```
void ExtractMotif(int p,  char A[], donor *motif)
{
        int first, last;
        int i, m;

        last = p + 5;
        first = p - 3;

        for (i = first; i <= last; i++)
        {
                m = i - first;

                switch (m)
                {
                        case 0:
                                motif->b1_8[0]
                                =motif->b1_7[0] = motif->b1_6[0] = A[i];
                                break;
                        case 1:
                                 motif->b1_8[1]
                                =motif->b1_7[1] = motif->b1_6[1]
                                =motif->b2_8[0] = motif->b2_7[0]
                                = motif->b2_6[0] = A[i];
                                break;
                        case 2:
                                 motif->b1_8[2]
                                =motif->b1_7[2] = motif->b1_6[2]
                                =motif->b2_8[1] = motif->b2_7[1]
                                =motif->b2_6[1]
                                =motif->b3_6[0] = motif->b3_7[0] =  A[i];
                                break;
                        case 3:
                                 motif->b1_8[3]
                                =motif->b1_7[3] = motif->b1_6[3]
                                =motif->b2_8[2] = motif->b2_7[2]
                                =motif->b2_6[2]
                                =motif->b3_6[1] = motif->b3_7[1]
                                =motif->b4_6[0] =  A[i];
                                break;
                        case 4:
                                motif->b1_8[4]
                                =motif->b1_7[4] = motif->b1_6[4]
                                =motif->b2_8[3] = motif->b2_7[3]
                                =motif->b2_6[3]
                =motif->b3_6[2] = motif->b3_7[2]
                =motif->b4_6[1] =  A[i];
                break;
            case 5:
                motif->b1_8[5]
                =motif->b1_7[5] = motif->b1_6[5]
                =motif->b2_8[4] = motif->b2_7[4]
                =motif->b2_6[4]
                =motif->b3_6[3] = motif->b3_7[3]
                =motif->b4_6[2] =  A[i];
                break;
            case 6:
                motif->b1_8[6]
                =motif->b1_7[6]
                =motif->b2_8[5] = motif->b2_7[5]
                =motif->b2_6[5]
                =motif->b3_6[4] = motif->b3_7[4]
                =motif->b4_6[3] =  A[i];
                break;
             case 7:
                motif->b1_8[7]
                =motif->b2_8[6] = motif->b2_7[6]
                =motif->b3_6[5] = motif->b3_7[5]
                =motif->b4_6[4] =  A[i];
                break;
            case 8:
                motif->b2_8[7]
                = motif->b3_7[6]
                =motif->b4_6[5] =  A[i];
                break;
        }  /*switch*/
    }
}

/*Write the donor data to a file*/
void WriteDonorToFile(char *fileName)
{
    int i;

    if ( (F_Out = fopen(fileName, "w")) == NULL)
    {
        printf("\nError: open %s file", fileName);
        exit(0);
    }
```

```
fprintf(F_Out, "\n\n**motif_b1_6**\n");
fprintf(F_Out, "Total:  %i\n", numb1_6);
for( i = 0; i < numb1_6; i++ )
        fprintf(F_Out, "\t%s\n",
                motif_b1_6[i]);

fprintf(F_Out, "\n\n**motif_b2_6**\n");
fprintf(F_Out, "Total:  %i\n", numb2_6);
for( i = 0; i < numb2_6; i++ )
        fprintf(F_Out, "\t%s\n",
                motif_b2_6[i]);

fprintf(F_Out, "\n\n**motif_b3_6**\n");
fprintf(F_Out, "Total:  %i\n", numb3_6);
for( i = 0; i < numb3_6; i++ )
        fprintf(F_Out, "\t%s\n",
                motif_b3_6[i]);

fprintf(F_Out, "\n\n**motif_b4_6**\n");
fprintf(F_Out, "Total:  %i\n", numb4_6);
for( i = 0; i < numb4_6; i++ )
        fprintf(F_Out, "\t%s\n",
                motif_b4_6[i]);

fprintf(F_Out, "\n\n**motif_b1_7**\n");
fprintf(F_Out, "Total:  %i\n", numb1_7);
for( i = 0; i < numb1_7; i++ )
        fprintf(F_Out, "\t%s\n",
                motif_b1_7[i]);

fprintf(F_Out, "\n\n**motif_b2_7**\n");
fprintf(F_Out, "Total:  %i\n", numb2_7);
for( i = 0; i < numb2_7; i++ )
        fprintf(F_Out, "\t%s\n",
                motif_b2_7[i]);

fprintf(F_Out, "\n\n**motif_b3_7**\n");
fprintf(F_Out, "Total:  %i\n", numb3_7);
for( i = 0; i < numb3_7; i++ )
        fprintf(F_Out, "\t%s\n",
                motif_b3_7[i]);

fprintf(F_Out, "\n\n**motif_b1_8**\n");
fprintf(F_Out, "Total:  %i\n", numb1_8);
```

```
        for( i = 0; i < numb1_8; i++ )
                fprintf(F_Out, "\t%s\n",
                        motif_b1_8[i]);

        fprintf(F_Out, "\n\n**motif_b2_8**\n");
        fprintf(F_Out, "Total:  %i\n", numb2_8);
        for( i = 0; i < numb2_8; i++ )
                fprintf(F_Out, "\t%s\n",
                        motif_b2_8[i]);

}


void InitiateScoreBoard()
{
        int i;
        for (i = 0; i < MaxSeq; i++ )
                ScoreBoard[i].Score = 0;
}

void VoteDonor(donor *D)
{
        int i;
        Score = 0;

        for (i = 0; i < numb1_6; i++ )
                if (strcmp(motif_b1_6[i], D->b1_6) == 0)
                        Score += point6;/*motif_b6.list[i].occur;*/

        for (i = 0; i < numb1_7; i++ )
                if (strcmp(motif_b1_7[i], D->b1_7) == 0)
                        Score += point7;

        for (i = 0; i < numb1_8; i++ )
                if (strcmp(motif_b1_8[i], D->b1_8) == 0)
                        Score += point8;


        for (i = 0; i < numb2_6; i++ )
                if (strcmp(motif_b2_6[i], D->b2_6) == 0)
                        Score += point6;

        for (i = 0; i < numb2_7; i++ )
                if (strcmp(motif_b2_7[i], D->b2_7) == 0)
                        Score += point7;
        for (i = 0; i < numb2_8; i++ )
                if (strcmp(motif_b2_8[i], D->b2_8) == 0)
                        Score += point8;

        for (i = 0; i < numb3_6; i++ )
                if (strcmp(motif_b3_6[i], D->b3_6) == 0)
                        Score += point6;

        for (i = 0; i < numb3_7; i++ )
                if (strcmp(motif_b3_7[i], D->b3_7) == 0)
                        Score += point7;

        for (i = 0; i < numb4_6; i++ )
                if (strcmp(motif_b4_6[i], D->b4_6) == 0)
                        Score += point6;
}



/*Input sequences from a file*/
void Classif(FILE *F_In, Grade ScoreBoard[])
{
        int i,n,p,
           seqlen; /*length of input seq*/
        int TempScor;
        char seq[MAXLENGTH],
                T[MAXLENGTH];

        for (TotalSeq =0; ;)
        {
                if ((fgets(T, MAXLENGTH, F_In)) == NULL)
                        break;
                if (T[0] == '>')
                        TotalSeq++;
        }
        printf("\nTotalSeq: %i\n", TotalSeq);
        if (TotalSeq > MaxSeq)
        {
                printf("\n%s\n", "Error: Too many sequences.");
                TotalSeq = MaxSeq;
        }
        rewind(F_In);
        for (n = 0; n < TotalSeq; n++)
        {
```

```
        if (fgets(T, MAXLENGTH, F_In) != NULL)
        {
            if (T[0] == '>')
            {
                n--;  /*do not count sequence name line*/
                continue;
            }
            i = 0;
            seqlen = strlen(T);
            if (T[seqlen -1] == '\n')
                seqlen--;

            strcpy(seq, T);
            seqlen = strlen(seq);
            TempScor = 0;

            for (i=3; i < seqlen -5 ; i++)
            {
                if ((seq[i] == 'G') && (seq[i+1] == 'T'))
                {
                    Score = 0;
                    p = i;
                    ExtractMotif(p, seq, &Donor);
                    VoteDonor(&Donor);

                    TempScor = Score;
                    if (TempScor > ScoreBoard[n].Score)
                        ScoreBoard[n].Score = TempScor;
                }
            }
        }
        if (ScoreBoard[n].Score >= Pos_Min)
            ScoreBoard[n].Class = Positive;
        else if (ScoreBoard[n].Score <= Neg_Max)
            ScoreBoard[n].Class = Negative;
        else
            ScoreBoard[n].Class = Unknown;
    }
}


void WriteScoreToFile(Grade A[], int Total, FILE *F_Out)
{
    int i;
```

```
          fprintf(F_Out, "\t Seq# \t   Score \t  Class\n\n");
          for (i = 0; i < Total; i++)
          {
                    fprintf(F_Out, "\t %3i \t %5i \t   %s\n", i+1,
                                    A[i].Score, A[i].Class);
          }
}
```

APPENDIX C

DONOR DETECTION PROGRAM

This is the Donor Site Detection program. This program can detect the real donor sites in the input DNA sequences. The Maximum DNA sequence is 5000 base, but it can be changed to different number. Please do not change other constant numbers.

```
/*Program Name: DonorDetection.c */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAXLENGTH 5000
#define TEMPLENGTH 300
#define MaxSeq 300


#define point6 64
#define point7 128
#define point8 256
#define point9 512
typedef struct _donor
        {
              char b1_6[7], b1_7[8], b1_8[9],
               b2_6[7], b2_7[8], b2_8[9],
               b3_6[7], b3_7[8],
               b4_6[7];
        } donor;

donor Donor;
typedef struct _spliceP
       {
              int Position;
              int Score;
              struct _spliceP *next;
       } spliceP;

typedef struct junctions
       {
              spliceP *Donor;
```

```
                spliceP *Acceptor;
        } Junctions;

Junctions *SpliceJunctions;

int TotalSeq;

int Score;

#define Pos_Min 768     /*Positive lower bound, inclusive*/
#define Neg_Max 640     /*Negative high bound*/
#define Positive "Donor"
#define Negative "Non-Donor"
#define Unknown "Unknown"

#define numb1_6 68
#define numb1_7 141
#define numb1_8 183
#define numb2_6 49
#define numb2_7 90
#define numb2_8 171
#define numb3_6 40
#define numb3_7 102
#define numb4_6 69
char
        motif_b1_6[numb1_6][7],
        motif_b1_7[numb1_7][8],
        motif_b1_7[numb1_7][8],
        motif_b1_7[numb1_7][8],
        motif_b1_8[numb1_8][9],
        motif_b2_6[numb2_6][7],
        motif_b2_7[numb2_7][8],
        motif_b2_8[numb2_8][9],
        motif_b3_6[numb3_6][7],
        motif_b3_7[numb3_7][8],
        motif_b4_6[numb4_6][7];

FILE  *Lib_In, *F_In, *F_Out, *Score_Out;

main()
{

        char sequence[50];
        char *donor_lib = "donordat.lib";
        char Score_file[50];
```

```
printf("\n\n\t\tWelcome for using this\n");
printf("\t\tDonor Detection program!\n");
printf("\n\t\tNote:");
printf("\n\t1) Please put the sequence file in the\n");
printf("\t   same directory as this program is in.\n");
printf("\t2) The max number of sequences is %i\n", MaxSeq);
printf("\t3) The results will be in the file name you specifed\n");

initiateData();
if ((Lib_In = fopen(donor_lib, "r")) == NULL)
{
        printf("ERROR: %s file open.\n", donor_lib);
        exit (0);
}


InportDonorLib();

fclose(Lib_In);


printf("\nPlease enter your sequence file:\n");
scanf("%49s", sequence);

if ((F_In = fopen(sequence, "r")) == NULL)
{
        printf("ERROR: %s file open.\n", sequence);
        exit (0);
}

Classif(F_In);
fclose(F_In);

printf("\nPlease enter your output file  to hold the results:\n");

scanf("%49s", Score_file);

if ((Score_Out = fopen(Score_file, "w")) == NULL)
{
        printf("ERROR: %s file open.\n", Score_file);
        exit (0);
}
WriteScoreToFile(TotalSeq, Score_Out);
```

```
    fclose(Score_Out);
    printf("\nThe program finished for Donor Recognition.\n");
    printf("Please look at your results in file: %s.\n",Score_file);

    return 0;
}



void initiateData()
{
    Donor.b1_6[6] = '\0';
    Donor.b1_7[7]= '\0';
    Donor.b1_8[8]= '\0';
    Donor.b2_6[6]= '\0';
    Donor.b2_7[7]= '\0';
    Donor.b2_8[8]= '\0';
    Donor.b3_6[6]= '\0';
    Donor.b3_7[7]= '\0';
    Donor.b4_6[6]= '\0';
}

void InportDonorLib()
{
    int i, j;
    char T[15],T1[5];
    for (i = 0; i < numb1_6; i++)
    {
        if (fgets(T, 10, Lib_In) == NULL)

        {
            printf("Error1: Donor Lib file open.\n");
            exit(0);
        }
        strcpy(T1, T);
        for(j = 0; j < 6; j++)
            motif_b1_6[i][j] = T1[j];
        motif_b1_6[i][6]='\0';

    }

    for (i = 0; i < numb2_6; i++)
    {
        if (fgets(T, 10, Lib_In) == NULL)
        {
                printf("Error2: Donor Lib file open.\n");
                exit(0);
        }
        strcpy(T1, T);
        for(j = 0; j < 6; j++)
                motif_b2_6[i][j] = T1[j];
        motif_b2_6[i][6]='\0';

}

for (i = 0; i < numb3_6; i++)
{
        if (fgets(T, 11, Lib_In) == NULL)
        {
                printf("Error3: Donor Lib file open.\n");
                exit(0);
        }
        strcpy(T1, T);
        for(j = 0; j < 6; j++)
                motif_b3_6[i][j] = T1[j];
        motif_b3_6[i][6]='\0';

}

for (i = 0; i < numb4_6; i++)
{
        if (fgets(T, 10, Lib_In) == NULL)
        {
                printf("Error: Donor Lib file open.\n");
                exit(0);
        }
        strcpy(T1, T);
        for(j = 0; j < 6; j++)
                motif_b4_6[i][j] = T1[j];
        motif_b4_6[i][6]='\0';
}


for (i = 0; i < numb1_7; i++)
{
        if (fgets(T, 10, Lib_In) == NULL)
        {
                printf("Error: Donor Lib file open.\n");
                exit(0);
        }
        strcpy(T1, T);
        for(j = 0; j < 7; j++)
                motif_b1_7[i][j] = T1[j];
        motif_b1_7[i][7]='\0';
}

for (i = 0; i < numb2_7; i++)
{
        if (fgets(T, 10, Lib_In) == NULL)
        {
                printf("Error: Donor Lib file open.\n");
                exit(0);
        }
        strcpy(T1, T);
        for(j = 0; j < 7; j++)
                motif_b2_7[i][j] = T1[j];
        motif_b2_7[i][7]='\0';
}

for (i = 0; i < numb3_7; i++)
{
        if (fgets(T, 10, Lib_In) == NULL)
        {
                printf("Error: Donor Lib file open.\n");
                exit(0);
        }
        strcpy(T1, T);
        for(j = 0; j < 7; j++)
                motif_b3_7[i][j] = T1[j];
        motif_b3_7[i][7]='\0';
}


for (i = 0; i < numb1_8; i++)
{
        if (fgets(T, 12, Lib_In) == NULL)
        {
                printf("Error: Donor Lib file open.\n");
                exit(0);
        }
        strcpy(T1, T);
        for(j = 0; j < 8; j++)
         motif_b1_8[i][j] = T1[j];
        motif_b1_8[i][8]='\0';
        }

        for (i = 0; i < numb2_8; i++)
        {
                if (fgets(T, 12, Lib_In) == NULL)
                {
                        printf("Error: Donor Lib file open.\n");
                        exit(0);
                }
                strcpy(T1, T);
                for(j = 0; j < 8; j++)
                        motif_b2_8[i][j] = T1[j];
                motif_b2_8[i][8]='\0';
        }

}

void ExtractMotif(int p,  char A[], donor *motif)
{
        int first, last;
        int i, m;

        last = p + 5;
        first = p - 3;

        for (i = first; i <= last; i++)
        {
                m = i - first;
                switch (m)
                {
                        case 0:
                                 motif->b1_8[0]
                                =motif->b1_7[0] = motif->b1_6[0] = A[i];
                                break;
                        case 1:
                                 motif->b1_8[1]
                                =motif->b1_7[1] = motif->b1_6[1]
                                =motif->b2_8[0] = motif->b2_7[0]
                                = motif->b2_6[0] = A[i];
                                break;
                        case 2:
                                motif->b1_8[2]
                                =motif->b1_7[2] = motif->b1_6[2]
                                =motif->b2_8[1] = motif->b2_7[1]
                                =motif->b2_6[1]
```

```
          =motif->b3_6[0] = motif->b3_7[0] =  A[i];
          break;
  case 3:
           motif->b1_8[3]
          =motif->b1_7[3] = motif->b1_6[3]
          =motif->b2_8[2] = motif->b2_7[2]
          =motif->b2_6[2]
          =motif->b3_6[1] = motif->b3_7[1]
          =motif->b4_6[0] =  A[i];
          break;
  case 4:
          motif->b1_8[4]
          =motif->b1_7[4] = motif->b1_6[4]
          =motif->b2_8[3] = motif->b2_7[3]
          =motif->b2_6[3]
          =motif->b3_6[2] = motif->b3_7[2]
          =motif->b4_6[1] =  A[i];
          break;
  case 5:
          motif->b1_8[5]
          =motif->b1_7[5] = motif->b1_6[5]
          =motif->b2_8[4] = motif->b2_7[4]
          =motif->b2_6[4]
          =motif->b3_6[3] = motif->b3_7[3]
          =motif->b4_6[2] =  A[i];
          break;
  case 6:
          motif->b1_8[6]
          =motif->b1_7[6]
          =motif->b2_8[5] = motif->b2_7[5]
          =motif->b2_6[5]
          =motif->b3_6[4] = motif->b3_7[4]
          =motif->b4_6[3] =  A[i];
          break;
    case 7:
          motif->b1_8[7]
          =motif->b2_8[6] = motif->b2_7[6]
          =motif->b3_6[5] = motif->b3_7[5]
          =motif->b4_6[4] =  A[i];
          break;
    case 8:
          motif->b2_8[7]
          = motif->b3_7[6]
          =motif->b4_6[5] =  A[i];
          break;
```

```
        } /*switch*/
    }
}

void VoteDonor(donor *D)
{
    int i;
    Score = 0;

    for (i = 0; i < numb1_6; i++ )
        if (strcmp(motif_b1_6[i], D->b1_6) == 0)
            Score += point6;/*motif_b6.list[i].occur;*/

    for (i = 0; i < numb1_7; i++ )
        if (strcmp(motif_b1_7[i], D->b1_7) == 0)
            Score += point7;

    for (i = 0; i < numb1_8; i++ )
        if (strcmp(motif_b1_8[i], D->b1_8) == 0)
            Score += point8;


    for (i = 0; i < numb2_6; i++ )
        if (strcmp(motif_b2_6[i], D->b2_6) == 0)
            Score += point6;

    for (i = 0; i < numb2_7; i++ )
        if (strcmp(motif_b2_7[i], D->b2_7) == 0)
            Score += point7;

    for (i = 0; i < numb2_8; i++ )
        if (strcmp(motif_b2_8[i], D->b2_8) == 0)
            Score += point8;

    for (i = 0; i < numb3_6; i++ )
        if (strcmp(motif_b3_6[i], D->b3_6) == 0)
            Score += point6;

    for (i = 0; i < numb3_7; i++ )
        if (strcmp(motif_b3_7[i], D->b3_7) == 0)
            Score += point7;

    for (i = 0; i < numb4_6; i++ )
        if (strcmp(motif_b4_6[i], D->b4_6) == 0)
```

```
                        Score += point6;
}

/*Input sequences from a file*/
void Classif(FILE *F_In)
{
        int i,n,p,j,m,
            seqlen; /*length of input seq*/
        int TempScor;

        char seq[MAXLENGTH] = "\0",
               T[TEMPLENGTH],
             T1[TEMPLENGTH];

        spliceP *tp1, *tp2;

        TotalSeq = 0;

        for (j =0; ;)
        {
                if ((fgets(T, TEMPLENGTH, F_In)) == NULL)
                        break;
                if (T[0] == '>')
                        TotalSeq++;
        }

        printf("\nTotalSeq: %i\n", TotalSeq);
        if (TotalSeq > MaxSeq)
        {
                printf("\n%s\n", "Error: Too many sequences.");
                TotalSeq = MaxSeq;
        }

        SpliceJunctions = malloc(TotalSeq*sizeof(Junctions));

        for ( j = 0; j < TotalSeq; j++)
        {
                SpliceJunctions[j].Donor = NULL;
                SpliceJunctions[j].Acceptor = NULL;
        }

        rewind(F_In);
        for (n = 0; n < TotalSeq; n++)
        {
                if (fgets(T, TEMPLENGTH, F_In) != NULL)
```

```
{
        if (T[0] == '>')
        {
                n--;  /*do not count sequence name line*/
                continue;
        }

        seqlen = strlen(T);

        m = 0;
        for (j = 0; j < seqlen; j ++)
        {
                if ((T[j] == 'A') || (T[j] == 'G') || (T[j] == 'C')
                 ||(T[j] == 'T') || (T[j] == 'N') )
                {
                        T1[m] = T[j];
                        m++;
                }
        }
        T1[m] = '\0';

        strcpy(seq, T1);

         for (i = 0; ;)
         {
                if (fgets(T, TEMPLENGTH, F_In) == NULL)
                break;
                if (T[0] == '>')
                        break;
                seqlen = strlen(T);

                m = 0;
                for (j = 0; j < seqlen; j ++)
                {
                        if ((T[j] == 'A') || (T[j] == 'G') || (T[j] == 'C')
                          ||(T[j] == 'T') || (T[j] == 'N') )
                        {
                                T1[m] = T[j];
                                m++;
                        }
                 }
                T1[m] = '\0';

                strcat(seq, T1);
          }
```

```
            seqlen = strlen(seq);
            if (seqlen > MAXLENGTH)
                seqlen = MAXLENGTH;
            TempScor = 0;

            for (i=3; i < seqlen; i++)
            {
                if ((seq[i] == 'G') && (seq[i+1] == 'T'))
                {
                    Score = 0;
                    p = i;
                    ExtractMotif(p, seq, &Donor);
                    VoteDonor(&Donor);

                    TempScor = Score;
                    if (TempScor > Pos_Min)
                    {
                        tp1 = malloc(sizeof(spliceP));
                        tp1->Position = i;
                        tp1->Score = TempScor;
                        tp1->next = NULL;
                        if (SpliceJunctions[n].Donor == NULL)
                        {
                            SpliceJunctions[n].Donor = tp1;
                            tp2 = SpliceJunctions[n].Donor;
                        }

                        else if (tp2->next == NULL)
                        {
                            tp2->next = tp1;
                            tp2 = tp2->next;
                        }
                        tp1 = NULL;
                    }
                }
            }
        }

    }
}


void WriteScoreToFile( int Total, FILE *F_Out)
{
    int i;
    spliceP *Tptr;

    fprintf(F_Out, "\t RESULTS\n");
    for (i = 0; i < Total; i++)
    {
        Tptr = SpliceJunctions[i].Donor;
        fprintf(F_Out, "\n  Sequence # %3i\n", i + 1);
        fprintf (F_Out, "\tPosition     Score\n");
        while (Tptr != NULL)
        {
            fprintf (F_Out, "\t%5i     %5i\n", Tptr->Position, Tptr->Score);
            Tptr = Tptr->next;
        }
    }
}
```

REFERENCES

1. Alberts, B., Bray, D., Lewis, J., Raff, M., Roberts, K. and Watson, J. D. *Molecular Biology of the Cell.* 3rd ed. Garland Publishing, Inc. New York and London, 1994.

2. Reed, R. and Maniatis, T. "Intron Sequences Involved in Lariat Formation during Pre-mRNA Splicing," *Cell,* 41, 95-105, 1985.

3. Treisman, R., Orkin, S.H. and Maniatis, T. "Specific Transcription and RNA Splicing Defects in Five Cloned β-Thalassaemia Genes," *Nature,* 302, 591-596, 1983.

4. Gelfand, M.S. "Prediction of Function in DNA Sequence Analysis," *J. Comput. Biol.,* 2(1), 87-115, 1995.

5. Iida, Y. "DNA Sequences and Multivariate Statistical Analysis. Categorical Discrimination Approach to 5' Splice Site Signals of mRNA Precursors in Higher Eukaryotes' Genes," *Comput. Appl. Biosci.* 3. 93-98, 1987.

6. Iida, Y. "Quantification Analysis of 5'-Splice Signal Sequences in mRNA Precursors. Mutations in Rabit β-Globin Gene," *Biochimica et Biophysica Acta*, 1007, 270-276, 1989.

7. Burset, M., and Guigo, R. "Evaluation of Gene Structure Prediction Programs," *Genomics*, 34, 353-367, 1996.

8. Marshall, E. "Emphasis Turns from Mapping to Large-scale Sequencing," *Science*, 268, 1270-1271, 1995.

9. Guigo, R.,Knudsen, S., Drake, N., and Smith, T.F. "Prediction of Gene Structure," *J. Mol. Biol.*, 226, 141-157, 1992.

10. Snyder, E. E., and Stormo, G. D. "Identification of Protein Coding Regions in Genomic DNA," *J. Mol. Biol.*, 248, 1-18, 1995.

11. Dong, S., and Searls, D.B. "Gene Structure Prediction by Linguistic Methods," *Genomics*, 23, 540-551, 1994.

12. Xu, Y., Mural, R.J., and Uberbacher, E.C. "Constructing Gene Models from Accurately Predicted Exons: An Application of Dynamic Programming," *Comput. Appl. Biosci.*, 10, 613-623, 1994.

13. Schneider, T. D., and Stephens, R. M. "Sequence Logos: a New Way to Display Consensus Structures," *Nucleic Acids Res.*, 18, 6097-6100, 1990.

14. Herman, N.D., and Schneider, T.D."High Information Conservation Implies that at Least Three Proteins Bind Independently to F Plasmid incD Repeats," *J. Bacteriol.*, 174, 3558-3560, 1992.

15. Papp, P.P., Chattoraj, D.K., and Schneider, T.D. "Information Analysis of Sequences that Bind the Replication Initiator RepA," *J. Mol. Biol.*, 233, 219-230, 1993.

16. Day, W.H.E., and McMorris, F.R. "Critical Comparison of Consensus Methods for Molecular Sequences," *Nucleic Acides Res.*, 20, 1093-1009, 1992

17. Staden, R. "Computer methods to Locate Signals in Nucleic Acid Sequences," *Nucleic Acids Res.*, 12, 505-519, 1984.

18. Shavlik, J.W., Towell, G.G., and Noordewier, M.O. "Using Knowledge-based Neural Networks to Refine Existing Biological Theories," in Lim, H.A., Fickett, J.W., and Robbins, R.J., eds., *Proc. 2nd Int. Conf. on Bioinformatics, Supercomputing and Complex Genome Analysis*, World Scientific, Singapore, 377-390, 1993

19. Milanesi, L., Kolchanov, N.A., Rogozin, I.B., Ischenlo, I.V., Kel, A.E., Orlov, Y.L., Ponomarenko, M.P., and Vezzoni, P. "GenViewer: A computing Tool for Protein-coding Regions Prediction in Nucleotide Sequences," in Lim, H.A., Fickett, J.W., Cantor, C.R., and Robbins, R.J., eds., *Proc. 2nd Int. Cong. On Bioinformatics, Supercomputing and Complex Genome Analysis,* World Scientific, Singapore, 573-587, 1993.

20. Lerner, M.R., Boyle, J. A., Mount, S. M., Wolin, S. L. and Steriz, J.A. "Are snRNPs involved in splicing?," *Nature*, 283. 220-224, 1980.

21. Mount, S. M. "A catalogue of splice junction sequences," *Nucleic Acids Res.*, 10, 459-472, 1982.

22. Kudo, M., Kitamura-Abe, S. Shimbo, M., and Iida, Y. "Analysis of context of 5'-splice site sequences in mammalian mRNA precursors by subclass method," *Comput. Appl. Biosci.*, 8. 367-376, 1992

23. Wang, J.T.L., Shapiro, B.A., and Shasha, D., edit., *Pattern Discovery in Molecular Biology.* Oxford University Press, New York, in progress.

24. Loewensterm, D., Hirsh, H., Yianilos, P., and Noordewier, M. "DNA Sequence Classification Using Compression-based Induction," *DIMACS Tech. Report*, Rutgers University, 1995.

25. Kudo, M., Iida, Y., and Shimbo, M. "Syntactic Pattern Analysis of 5'-splice Site Sequences of mRNA Precursors in Higher Eukaryotic Genes," *Comput. Appl. Biosci.*, 3. 319-324, 1987