BIO-PROGRAMMING PROSPECTS OF JAVA: A COMPUTATIONAL MOVE TOWARDS THE UNDERSTANDING OF THE BIOLOGICAL ASPECTS OF GENES AND PROTEINS

KALPANA RAJA

Data Mining and Text Mining Laboratory, Department of Bioinformatics, Bharathiar University, Coimbatore, Tamilnadu -641046, India
kalpana.rajaa@gmail.com

Abstract
Java is a powerful object oriented programming language that dominates many other programming languages for more than a decade. It is well designed and available as many executable technologies for software development such as Java Swing, Java Beans, Java Applets, Java Web Start, Java Database Connectivity (JDBC), Java Server Pages (JSP) and Java 2 Enterprise Edition (J2EE). Beyond its usage in the IT sector, the language is prominent even in the new emerging fields including bioinformatics and computational biology. The biological data (genes and proteins) from the biological and medical research is immense and require software professionals to mine them for new knowledge discovery. The knowledge to merge the programming concepts of Java to understand a wide range of biological concepts opens a new career challenge for many IT professionals. This paper introduces the implementation of the coding knowledge of Java in the field of molecular biology.

Keywords: Bio-programming; Pattern matching; Bioinformatics.

1. Introduction

Java has been the language of choice for beginning computing science courses at many universities and for many companies in projects and products development. It is a very well designed language as well as a pleasure to program in. Java is a powerful object oriented programming language that dominates many other programming languages for more than a decade. Java evolves many of the object oriented programming concepts from C++, a direct descendent language of C. Even though it derives many of its features from C and C++, it has many significant practical and philosophical differences [1]. The language is platform independent or architecture-neutral and supports the development of both online and offline applications. It is also well known for security and portability. Java achieves this protection by confining a Java program to the Java execution environment and not allowing its access to other parts of the computer. Likewise, it supports the portability of executable code in internet where many types of computers and operating systems are in use throughout the world. In other words, Java's solution for the two major problems namely, security and portability is both elegant and efficient. The key achieving the mentioned problems is that the output of a Java compiler is not the executable code, but the byte codes. During execution, the Java run-time system called Java virtual machine (JVM) interprets the byte code to produce the output [2]. The major advantages of developing software using Java include simple, object-oriented, robust, multithreaded, high performance, distributed and dynamic way of programming. Some of the well-known executable technologies of Java available for software development include Java Swing, Java Beans, Java Applets, Java Web Start, Java Database Connectivity (JDBC), Java Server Pages (JSP) and Java 2 Enterprise Edition (J2EE) [3].

Apart from the common software development, Java has intruded into the computational design and development of many core concepts of molecular biology. The bioinformatics and computational biology are the two major research fields merging the concepts of biology and software programming. A major driving force behind the research in biology and medicine over the past decade or two is due to the availability of huge amounts of DNA and protein data. The Human Genome Project is one of the major initiatives [4]. It led to massive improvements in the efficiency of DNA sequencing, and inspired numerous other genome

projects. National Center for Biotechnology Information (NCBI) [5] and National Institute of Health (NIH) at US [6], European Bioinformatics Institute (EBI) at UK [7], European Molecular Biology Laboratory (EMBL) at Germany [8] and Swiss Institute of Bioinformatics (SIB) [9] at Switzerland are the chief organizations involved in genome research. Besides, many small organizations are building software portals to collaborate with one another to share data and research results. A complete biological application depends on five basic concepts: (i) a source of data, (ii) an application programming language to access and analyze the data, (iii) reuse software tools and libraries developed by others (iv) a web application platform to provide a HTML user interface for the data and analysis results and finally (v) a data store, such as a relational database, to store results or user's data. Here the latter two notions are optional, the middle one is highly desirable and the first two are considered to be mandatory [10].

Among the various programming languages, Java has been widely accepted for biological software development. The concepts of Java are developed with code reuse in mind and the regular expression library of Java is not part of the basic syntax of the language but vast and defined tangibly [11]. It is more verbose than Perl, an equally accepted programming language for developing biological software. This paper introduces the Java programming perceptions in the field of biology and medicine together with a basic understanding of genes and proteins.

2. Bio-programming for protein synthesis – Transcription and Translation

2.1. Central Dogma of Molecular Biology

The flow of genetic information from DNA -> RNA -> protein in the biological systems is explained by the "Central Dogma of Molecular Biology" [12]. DNA (deoxyribonucleic acid) encodes the genetic information for most species. In order to use this information to produce proteins, the DNA must first be converted into a messenger RNA (ribonucleic acid) through a process called transcription. The information carried by the mRNA is then used to construct a specific protein (or polypeptide) through a process called translation (Fig 1). *In-vitro* process of understanding the protein synthesis is both complicated and time-consuming. The application of computer technology to study such biological concepts has opened up a new dimension of Java programming to develop many specialized biological software.

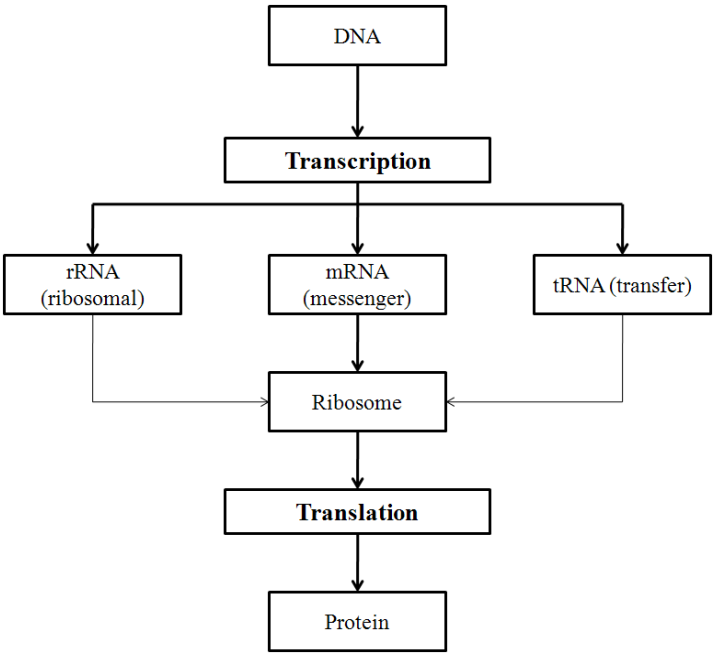

Fig 1. Flow of genetic information from DNA -> RNA -> Protein

2.1.1. Transcription of DNA sequence to RNA sequence

Gene expression is the process whereby a gene's DNA is used for the direct synthesis of a specific protein. The information encoded in a specific region of DNA is transcribed (copied) to produce a specific molecule of RNA. During transcription, genetic information in DNA serves as a template for copying the information into a complementary sequence of RNA. Three kinds of RNA made from the DNA template are messenger RNA (mRNA), ribosomal RNA (rRNA) and transfer RNA (tRNA). The enzyme RNA polymerase catalyzes the transcription of DNA [13]. While transcribing, the three nucleotide bases (Thymine (T), Guanine (G), Cytosine (C)) in the DNA template pair in a complementary manner with Adenine (A), C and G, respectively in the RNA strand. However, A in the DNA template pairs with Uracil (U), not T in RNA (Fig 2). The corresponding algorithm and java code for converting DNA sequence to RNA sequence are presented in Fig 3.

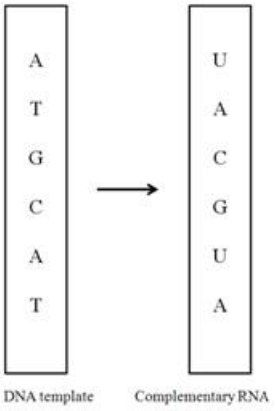

Fig 2. Base-pairing of nucleotides

Algorithm

- Start execution
- Read DNA sequence
- Split bases into character array
- Loop
  - Change base 'c' to 'g'
  - Change base 'g' to 'c'
  - Change base 't' to 'a'
  - Change base 'a' to 'u'

  End Loop
- Convert character array to string, RNA
- Print RNA sequence
- End execution

Java code

```
public class TranscribeDNAtoRNA  {
   public static void main(String[] args) {
      try {
         String dna = "atgccgaatcgtaa";
         char[] readdna;

         readdna = dna.toCharArray();

         for(int x=0; x<readdna.length; x++) {
            if (readdna[x] == 'c')
               readdna[x] = 'g';
            else if (readdna[x] == 'g')
               readdna[x] = 'c';
            else if (readdna[x] == 't')
               readdna[x] = 'a';
            else if (readdna[x] == 'a')
               readdna[x] = 'u';
         }

         String rna = new String(readdna);

         System.out.println(rna);
      } catch (Exception e) {
         System.err.print(e);
      }
   }
}
```

Fig 3. Bio-programming for transcription process in protein synthesis

2.1.2. Translation of RNA sequence to protein

RNA that transcribed from DNA then attaches to a ribosome, where the information contained in RNA is translated into a corresponding sequence of amino acids to form a new protein molecule. The translation process begins at the start codon (AUG) on mRNA and runs until any one of the stop codons (TAG, TGA, TAA) hit [13]. Codon is the transcribed base triplet coding for a particular amino acid. For example, the base triplet AUG codes for the amino acid methionine. In other words, methionine is always the first amino acid in a growing polypeptide. For detailed information in codons and aminoacids, refer [14]. The algorithm involved in the translation process of RNA sequence to protein is much more complicated than the transcription process (Fig 4).

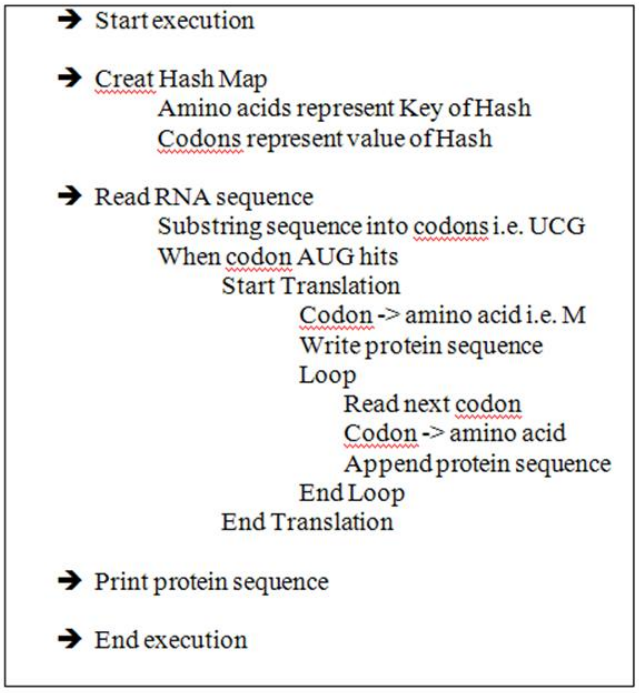

Fig 4. Bio-programming algorithm for translation process in protein synthesis

2.1.2.1. Java code to translate RNA sequence to protein

Below is the complete Java code meant to translate RNA sequence to protein. The program implements Hash MAP to store the amino acids and equivalent codons as key and values, which are later utilized to translate the RNA sequence to protein.

```
import java.util.HashMap;
import java.util.Map;

public class Translate {
        // RNA String to translate
        private static final String RNA = "auggcacaggcacuguugguacc";

        // Map to store the codons to an amino acid sequence
        private static final Map<String ,String> TRANSLATION = new HashMap<String ,String>();

        // Codons to translate
        private static final String[] CODONS = { "uuu", "uuc", "uua", "uug",
                                                 "ucu", "ucc", "uca", "ucg",
                                                 "uau", "uac", "uaa", "uag",
```

```
                                                "ugu", "ugc", "uga", "ugg",
                                                "cuu", "cuc", "cua", "cug",
                                                "ccu", "ccc", "cca", "ccg",
                                                "cau", "cac", "caa", "cag",
                                                "cgu", "cgc", "cga", "cgg",
                                                "auu", "auc", "aua", "aug",
                                                "acu", "acc", "aca", "acg",
                                                "aau", "aac", "aaa", "aag",
                                                "agu", "agc", "aga", "agg",
                                                "guu", "guc", "gua", "gug",
                                                "gcu", "gcc", "gca", "gcg",
                                                "gau", "gac", "gaa", "gag",
                                                "ggu", "ggc", "gga", "ggg"
    };

    // Amino acid in map
    private static final String[] AMINO_ACIDS = { "F", "F", "L", "L",
                                                  "S", "S", "S", "S",
                                                  "Y", "Y", "--STOP--", "--STOP--",
                                                  "C", "C", "--STOP--", "W",
                                                  "L", "L", "L", "L",
                                                  "P", "P", "P", "P",
                                                  "H", "H", "Q", "Q",
                                                  "R", "R", "R", "R",
                                                  "I", "I", "I", "M",
                                                  "T", "T", "T", "T",
                                                  "N", "N", "K", "K",
                                                  "S", "S", "R", "R",
                                                  "V", "V", "V", "V",
                                                  "A", "A", "A", "A",
                                                  "D", "D", "E", "E",
                                                  "G", "G", "G", "G"
    };

    // initialize the map
    private static void init() {
          for (int i=0; i<CODONS.length; i++)
              TRANSLATION.put(CODONS[i], AMINO_ACIDS[i]);
    }

     public static void main(String[] argv) {
           init();
           StringBuffer aminoAcidSequence = new StringBuffer();
           int i = 0;
           while (i < RNA.length()) {
               aminoAcidSequence.append (TRANSLATION.get(RNA.substring(i, i+3)));
                i += 3;
           }
           System.out.println("Amino acid string: " + aminoAcidSequence);
     }
    }//end class Translate
```

3. Pattern matching with Java

In general, pattern matching is the technique of searching a string containing text or binary data for some set of characters based on a specific search pattern. It is more than just searching for some set of characters in a data; it is a way of looking and processing a data in a manner that can be incredibly efficient and amazingly easy to program. The importance of this technique is evident from its application in various fields, not limiting to parsers, spam filters, digital libraries, screen scrapers, word processors, web search engines, natural language processing, feature detection in digitized images and many more [15]. In computational molecular biology, pattern matching is applicable to a variety of research areas such as optimal sequence alignment, multiple sequence alignment, and building models to describe the sequence families using Hidden Markov Models (HMMs) and regular expressions [16].

3.1. Sequence alignment

DNA and protein sequences determine the structure which in turn recognizes the function. The sequence alignment is a way of arranging the sequences of DNA, RNA, or protein to identify regions of similarity that may be a consequence of functional, structural, or evolutionary relationships between the sequences. Aligned sequences have two aspects; (i) quality or the region of similarity between the sequences and (ii) quantity or the degree of similarity. The two classical types of sequence alignment are pair-wise alignment (between two sequences) and multiple sequence alignment [17]. A simple algorithm compares every character (nucleotide in DNA sequence and amino acid in protein sequence) between the two sequences and indicates the matches (Fig 5A). Advanced programming algorithms involve scoring systems using various sequence analysis matrices like BLOSUM50 substitution matrix and PAM.

3.2. Conserved regions

In biology, conserved sequences are similar or identical sequences that occur within nucleic acid sequences (such as RNA and DNA sequences), protein sequences, protein structures or polymeric carbohydrates across the species (*orthologous* sequences) or within different molecules produced by the same organism (*paralogous* sequences). In the case of cross species conservation, a particular sequence may have been maintained by evolution despite speciation. Highly conserved proteins are often required for basic cellular function, stability or reproduction. Among the most highly conserved sequences are the active sites of enzymes and the binding sites of protein receptors [17]. Identifying the conserved regions in DNA or protein sequences is a core area in bioinformatics research and is possible through regular expressions of Java. Java provides the *java.util.regex* package for pattern matching with regular expressions. Java regular expressions are very simple to learn and use once the required biological concept is understood clearly (Fig 5B).

A

```
public class AlignSequences {
   public static void main(String[] argv) {
      try {
         String seq1, seq2;
         int len, i;

         seq1 = "YECNERSKAFSCPSHL";
         seq2 = "YECNQCGKAFAQHSSL";

         len = seq1.length();

         System.out.println(seq1);
         System.out.println(seq2);

         while (i<len) {
            if (seq1.charAt(i) ==
                                 seq2.charAt(i))
               System.out.print("*");
            else
               System.out.print(" ");
            i++;
         }
      } catch (Exception e) {
         System.err.println(e);
      }
   }
}
```

B

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class SeqPatternMatch {
   public static void main(String[] args) {
      String seq = "MAVMEFVCVRQKV");
      String regEx = "[A|C].V..[^ED]";

      Pattern p = Pattern.compile(regEx,
                  Pattern.CASE_INSENSITIVE);

      Matcher m = p.matcher(seq1);

      while(m.find()) {
        System.out.println (
                     "Pattern found @ start:" +
                     m.start() +
                     " and end positions:" +
                     m.group());
      }
   }
}
```

Fig 5. A. Bio-programming for sequence alignment. B. Bio-programming for conserved regions

4. Java APIs for Bioinformatics

Java provides a rich source of APIs for bioinformatics study of DNA and protein sequences. However, design and maintenance of these APIs present their own challenges [18]. The development of APIs that couple biological and computational knowledge to formally describe complex biological data types significantly reduces the number of conflicting formats and the time required to access and meaningfully analyzes biological data. Some of the Java-based bioinformatics APIs that are being developed to reduce the programming task are listed below.

- **BioJava** - An open source project dedicated to provide Java tools for processing biological data. BioJava's goal is to create an API that automates common bioinformatics tasks while providing a foundation for bioinformatics-based software projects.
- **caBio (Cancer Bioinformatics Infrastructure Objects)** is one component of the National Cancer Institute's Centre for Bioinformatics (NCICB), caCORE research management system. The caBio API contains the implementations of various biomedical objects to facilitate consistent data representation and data integration projects.
- **ENSJ** is the Java implementation of the EnsEMBL driver and data adaptors. ENSJ allows a developer to access sequence or annotation information stored in the EnsEMBL database. Recently, a new prototype API, called MartJ, has been developed and allows the developers to access EnsEMBL's Mart database that focuses on the fast and flexible multi-organism data-mining.
- **Phylogenetic Analysis Library (PAL)** is a Java API dedicated to the subset of bioinformatics analysis that pertains to the evolutionary development of genomes (DNA and protein sequence).

- **KDOM -** The Knowledge Discovery Object Model (KDOM) is a bioinformatics-based API designed to represent and manage biological knowledge during application development.
- **MAGE-stk** is an example of a bioinformatics-based API that provides a Java representation of the information required to describe a particular experiment such as a microarray experiment.

5. Conclusions

Java is one of the best suited programming languages to understand the hidden knowledge of biology. The Java resources (packages and APIs) available for bioinformatics and computational biology are vast, opening a new dimension of career aspects for many programmers. The amount of biological data needed to be mined is enormous and require many computer experts to work on. Basic knowledge on DNA and protein sequences along with their coding possibilities using Java can bring in new knowledge in many scientific research areas such as genomics, drug design, pharmacogenomics and cheminformatics.

Acknowledgment

The periodic help rendered by Mrs. V. Srividhya, Department of Bioinformatics, Centre for Plant Molecular Biology, Tamil Nadu Agricultural University, Coimbatore, in understanding the biological concepts is acknowledged.

References

[1] Schildt H. The complete reference – Java 2 Seventh Edition
[2] H. M. Deitel - Deitel & Associates, Inc., P. J. Deitel - Deitel & Associates, Inc. JavaTM How to program.
[3] Sixth Edition
[4] Alain Trottier. Java 2 Core Language Little Black Book
[5] Human Genome Project (HGP),
[6] http://www.ornl.gov/sci/techresources/Human_Genome/project/about.shtml (13 Nov 2011)
[7] National Center for Biotechnology Information (NCBI), http://www.ncbi.nlm.nih.gov/ (13 Nov 2011)
[8] National Institute of Health (NIH), http://www.nih.gov/ (13 Nov 2011)
[9] European Bioinformatics Institute (EBI), http://www.ebi.ac.uk/ (13 Nov 2011)
[10] European Molecular Biology Laboratory (EMBL), http://www.embl.de/ (13 Nov 2011)
[11] Swiss Institute of Bioinformatics (SIB), http://www.isb-sib.ch/ (13 Nov 2011)
[12] Young, E.; Alper, H. (2009): Synthetic Biology: Tools to Design, Build, and Optimize Cellular Processes,
[13] Journal of biomedicine and biotechnology, 2010, 1-12.
[14] Regular Expressions in Java, http://www.tutorialspoint.com/java/java_regular_expressions.htm (13 Nov
[15] 2011)
[16] Crick, F. (1970): Central Dogma of Molecular Biology, Nature, 227, 561-563.
[17] Skinner, M. (2011): Protein synthesis: An expressive couple, Nature Reviews Molecular Cell Biology,
[18] 12(8).
[19] DNA & RNA Codons, http://molvis.sdsc.edu/dna/codons.htm (13 Nov 2011)
[20] Runar. (2009): Structural Pattern Matching in Java, http://apocalisp.wordpress.com/ 2009/08/21/ structural-
[21] pattern-matching-in-java/ (13 Nov 2011)
[22] Rouchka, E. C. (1999): Pattern Matching Techniques and Their Applications to Computational Molecular
[23] Biology - A Review, bioinformatics.louisville.edu/localresources/papers/WUCS-99-09.pdf (13 Nov 2011)
[24] Durbin, R.; Eddy, S.; Krogh, A.; Mitchison, G. Biological sequence analysis: Probabilistic models of
[25] proteins and nucleic acids.
[26] Montgomery, S. (2004): Java APIs for Bioinformatics,
[27] http://onjava.com/pub/a/onjava/2004/03/10/bioinf.html (13 Nov 2011)