APPLICATION OF COMPUTATIONAL METHODS TO STUDY THE SELECTION OF AUTHENTIC AND CRYPTIC SPLICE SITES

A Project Report
Presented to
Dr. Sami Khuri
Department of Computer Science
San Jose State University

In Partial Fulfillment
of the Requirements for the Class
CS 298

By

Tapomay Dey

April 2017

ABSTRACT

This report explains the significance of the process of splice site selection during the translation of pre-mRNA into proteins. This sets the premise for a very interesting and highly valuable problem of putative splice site detection. Our eventual goal is to better understand the mechanism by which cryptic splice sites are activated in presence of mutations at authentic splice sites. We try to address the research question whether we can predict putative splice sites using probabilistic models. This report explains the following three important machine learning algorithms: Decision trees, Random Forests, and Hidden Markov Models. The report also touches very lightly on recent advances in application of evolutionary algorithms to the general problem of splice site detection. The last section presents a hypothetical and intuitive analogy of how one may model the same problem using Convolutional Neural Network, a variant of Artificial Neural Network that is highly accurate in image recognition.

Contents

1 MOTIVATION

The eukaryotic genome consists of many genes. Transcription and translation are some of the important processes in the gene expression pathways. Transcription involves generation of messenger RNA (mRNA) from pre-messenger RNA (pre-mRNA). Both mRNA and pre-mRNA are sequences of nucleotides. The pre-mRNA is divided into nucleotide regions called Introns and Exons. An Exon-Intron boundary is called a 5' splice site and an Intron-Exon boundary is called a 3' splice spite. As per recent knowledge, the Introns are discarded and the Exons are stitched together to form the mRNA.

The mRNA is the exact coding sequence(CDS) that is translated into proteins. Proteins comprise of sequences made from various combinations of twenty possible amino acids. Each of the twenty amino acids can be mapped to multiple nucleotide triplets (codons) from the mRNA. One of the first publications of these mappings was by Crick in The Origin of the Genetic Code [1]. The exact sequence of amino acids dictates the structural properties of the protein molecule. The structure and composition of a protein molecule influence the physical and chemical properties exhibited by the protein. These properties have been documented in the Wikipedia page on Amino Acids [2].

The proteins are responsible for various biological functions and traits of a eukaryotic organism. The splicing of pre-mRNA at 5' and 3' splice sites must be accurate in order to form the protein molecules that are required

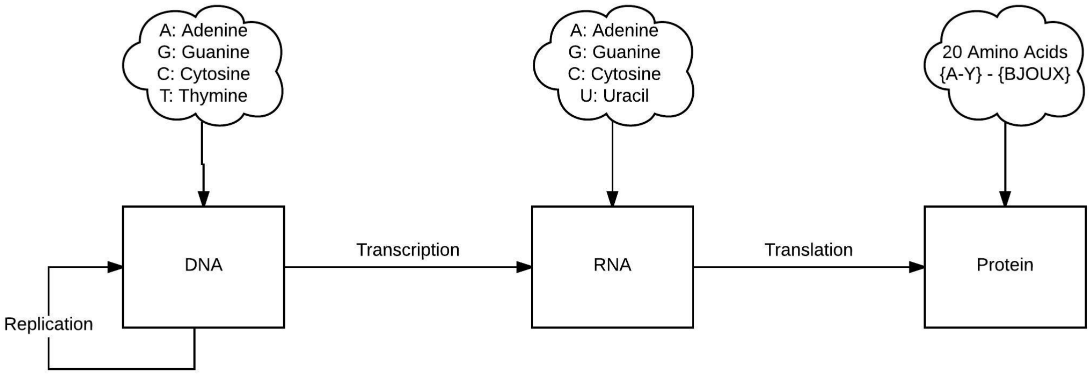


Figure 1: Protein synthesis

for satisfying the normally known functions of a healthy organism.

Mutations in the DNA can lead to one of many possible disruptions in the selection of 5' or 3' splice sites. An obviously devastating disruption is a frame shift while extracting codons from the mRNA to form amino acid molecules. This may lead to a severely deficient expression of an important protein due to early detection of stop codons. In many cases, this leads to undesired consequences like fatal diseases. As per recent estimates, it is known that up to 50% of disease-causing mutations disrupt splicing.

The splice sites that are selected due to mis-splicing are called cryptic splice sites. However, there are multiple nucleotide subsequences in the pre-mRNA that contain candidate splice sites. Consequently, it is of crucial importance to understand the reasons behind the cryptic splice site selection by the spliceosome.

2 PROBLEM STATEMENT

To study the selection of splice sites by the spliceosome, three data sets were built, consisting of authentic, cryptic, and neighboring 5' splice sites. The data sets comprise of thousands of 9-mers: sequences that are 9 bases long. Authentic and cryptic splice sites are extracted from public datasets. The neighboring splice site 9-mers are extracted from 100 base-pairs downstream and 100 base-pairs upstream of each cryptic splice site.

The primary goal is to build probabilistic models for each data set and quantitatively compare their similarities and differences. The primary hypothesis is that the authentic and cryptic splice sites are inherently different. The secondary hypothesis is that the neighboring splice sites are more dissimilar than both authentic and cryptic splice sites. This should help us understand the behavior of a spliceosome when it chooses cryptic splice sites and discards neighboring splice sites in case of mutations that disrupt the authentic splice site.

3 DATA SPECIFICATIONS

3.1 5' SPLICE SITES

The 5' splice sites are extracted based on the location of the invariant GT dinucleotide. The authentic splice sites are extracted from the Homo Sapiens Splice Sites database (HS3D) [?, 4]. The cryptic splice sites and neighboring splice sites are extracted from the DBASS5 database [5, 8].

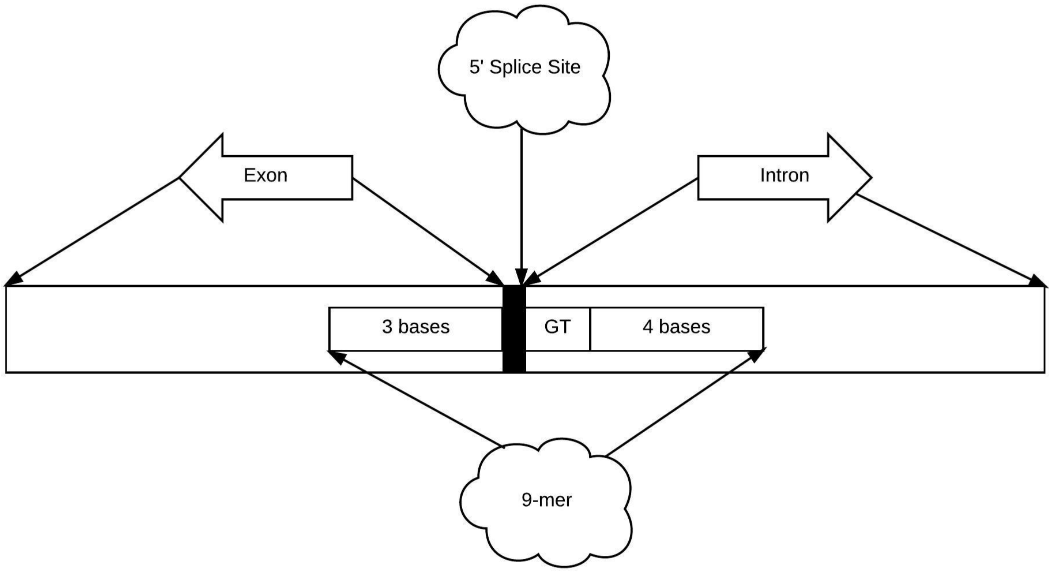
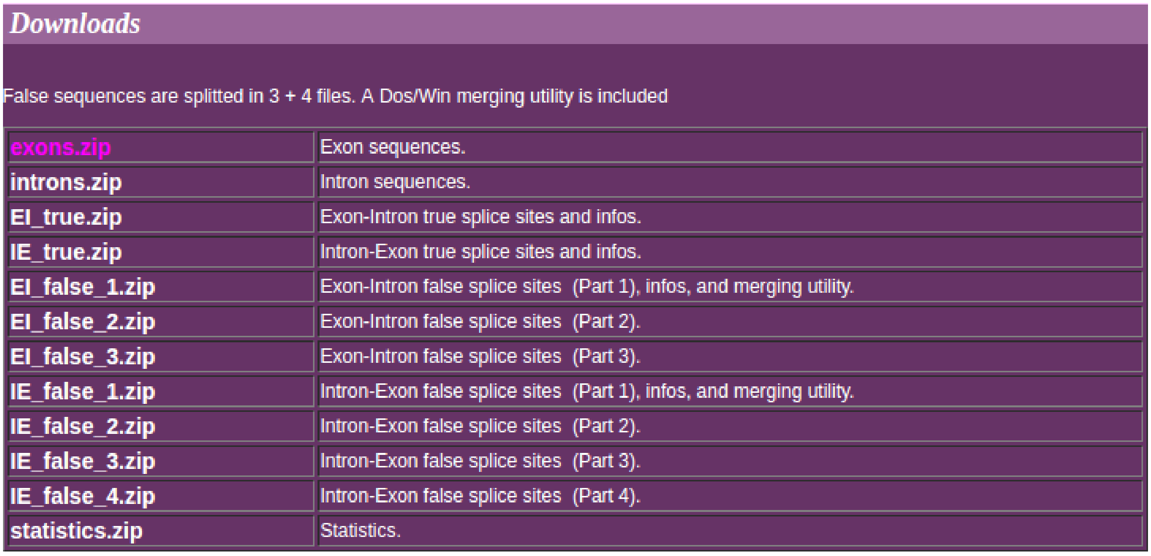


Figure 2: 5’ Splice Site

3.2 AUTHENTIC 5’ SPLICE SITE

HS3D offers the following files listed in figure-3 for download:



Downloads

False sequences are splitted in 3 + 4 files. A Dos/Win merging utility is included

exons.zip	Exon sequences.
introns.zip	Intron sequences.
EI_true.zip	Exon-Intron true splice sites and infos.
IE_true.zip	Intron-Exon true splice sites and infos.
EI_false_1.zip	Exon-Intron false splice sites (Part 1), infos, and merging utility.
EI_false_2.zip	Exon-Intron false splice sites (Part 2).
EI_false_3.zip	Exon-Intron false splice sites (Part 3).
IE_false_1.zip	Intron-Exon false splice sites (Part 1), infos, and merging utility.
IE_false_2.zip	Intron-Exon false splice sites (Part 2).
IE_false_3.zip	Intron-Exon false splice sites (Part 3).
IE_false_4.zip	Intron-Exon false splice sites (Part 4).
statistics.zip	Statistics.

Figure 3: HS3D downloads page

Since 5’ splice sites occur at the Exon-Intron boundary, the file EI_true.zip is relevant for Authentic 5’ splice sites. The data is provided as an ‘seq’ file with 140 nucleotides in each line. Manually observing all the sequences reveals that the 5' consensus GT dinucleotide that marks the start of the Intron is at position 71-72 in each sequence. The desired 5' splice site 9-mer comprises of the last 3 nucleotides from the Exon, the GT dinucleotide from the start of Intron and the 4 nucleotides following it.

Following is a sample of 9-mers extracted from the dataset:

Table 1: Input sequence lines from file EI_true.seq

AB000381 (1,2,2): CTCCTCTTTGCCTTACTCCTAGCCATGGAGCTC CCATTGGTGGCAGCCAGTGCCACCATGCGCGCTC**AGTGTAAGT** ATCATTCCCTCTCACTGTCCTGGAGAGGACGAGAATTCCACCT GGGGTGCTGGGGGTCACTGGG
AB000381 (2,3,3): AATGACTTCAACTGTCCCAACATTAGAGTATGT CCGTATCATATTAGGCGCTGTATGACAATCTCCA**TTCGTAAGT** ACCTCTTGGTCATTTGGACACATTGTAGATTAGTCCCCTACCT GGGTAGTTTCTGGGGCCAGGG
AB002059 (3,1,1): TGACCAGGAACTGGCGGGTGGGCGCCCTGCAGA GGCTGCTGCAGTTTGGGATCGTGGTCTATGTGGT**AGGGTAAGA** GAGAAGAGCTTTTGGCCAGGCTGGAGGGGCAAGGGAAGAGGTG GGGGGTGGGGCTTGGTCCTGC
AB002059 (4,2,2): TTCCGTCACTCAGATCAAGGAGCTTGGAAACCG GCTGTGGGATGTGGCCGACTTCGTGAAGCCACCT**CAGGTGGGG** GCCCTGATGTTGCTGACGGGGGCGCAAGTCCTTTCCCCACTGA CAGCCTGAACACCCGCCATGC

Table 2: 9-mers extracted from the sequences in Table 1

AGT**GT**AAGT
TTC**GT**AAGT
AGG**GT**AAGA
CAG**GT**GGGG

Total authentic 5' splice site 9-mers extracted : **2796**

3.3 CRYPTIC 5' SPLICE SITE

The cryptic 5' splice sites are collected from the DBASS5 database portal [5, 8] by crawling all available splice details page. The splice site details page contains the nucleotide sequence with the context of the mutation that alters the splicing location. The mutation is indicated in the sequence with a greater than symbol.

For example: (G>A) means G is replaced by A

The cryptic splice sites are denoted by the slash symbol.

Consider the following nucleotide sequence extracted from the first record.

URL: http://www.dbass.org.uk/DBASS5/viewsplicesite.aspx?id=627 (last retrieved: 04/23/2017)

CCAGCAACTT GGCTCTTTTT GGAGAGCGGC TGGGCCTGGT TGGCCACAGC CCCAGTTCTG CCAGCCTGAA CTTCCTCCAT GCCCTGGAGG TCATGTTCAA ATCCACCGTC CAGCTCATGT TCATGCCCAG GAGCCTGTCT CGCTGGACCA GCCCCAA**G/G** TGTGGAAGGA GCACTTTGAG GCCTGGGACT GCATCTTCCA GTAC**(G>A)**g tgaggccagg gacccgggca gtgctatggg gaagggacac catgggggcc caatttctcc ctctccacca cccagtgggg aatggaggcc acagggaggg gtcggggatt cctcaccttc ctgccaggga gattggtgcg aggctggggc tgggctgggc tgatccggag aatttgggat gagagcaggg agacttgggt gtcggggcag tctgggcagg aggaggacac tgaaggatgt ctcccagcac caaagtctga gggctgcctc ccgctccccg

We can observe from other details on the page that:

- Capital nucleotide symbols indicate the trailing part of the Exon
- Small nucleotide symbols indicate the leading part of the Intron
- Mutation is marked as : (G>A)
- Cryptic splice site is marked as : “GCCCCAAG/G TGTGGAAGG”

The cryptic 5’ splice site 9-mer extracted from the above sequence is: AAGGTGTGG

Total cryptic 5’ splice site 9-mers extracted : **539**

3.4 3’ SPLICE SITE

The 3’ splice sites are extracted based on the location of the invariant AG dinucleotide and its preceeding Yn consensus were Y is a Pyridine. The authentic splice sites are extracted from the Homo Sapiens Splice Sites database

(HS3D) [3, 4]. The cryptic splice sites and neighboring splice sites are extracted from the DBASS3 database [5–7].

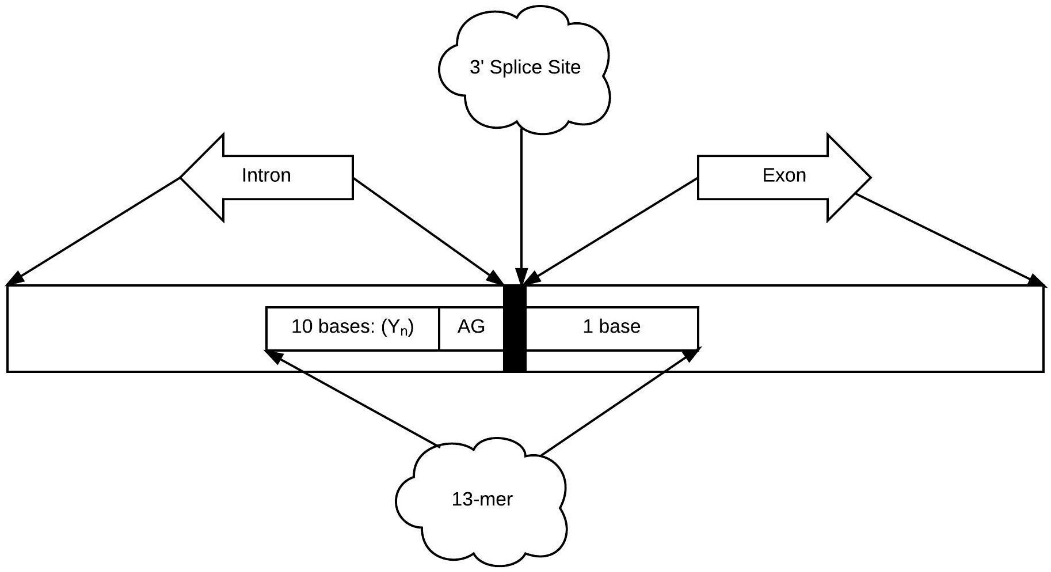


Figure 4: 3’ Splice Site

3.5 AUTHENTIC 3’ SPLICE SITE

Since 3’ splice sites occur at the Intron-Exon boundary, the file IE_true.zip is relevant for Authentic 3’ splice sites. This data is also provided as an ‘seq’ file with 140 nucleotides in each line. Manually observing all the sequences reveals that the 3’ consensus AG dinucleotide that marks the end of the Exon is at position 69-70 in each sequence. The desired 3’ splice site 13-mer comprises of the last 12 nucleotides from positions 59-70 at the end of the Exon, and one nucleotide from the start of the Intron.

Following is a sample of 13-mers extracted from the dataset:

Table 3: Input sequence lines from IE_true.seq

AB000381(1,1,2): GGCCAGGGGCATAGAGCTGGCCAAGGAGCCATGGCTCAC TAACGTGTTGTATGGGGCT**CCTTCCCTTCAGG**TCCAGGCTCCTGCGTGAAG TGATGCTCCTCTTTGCCTTACTCCTAGCCATGGAGCTCCCATTGGTGGCA
AB000381(2,2,3): GAGTGAGCTGGTAATGGGTGGAAAAGGCGTAGTGGAGCA GAAGCCTGAAGCCTGCTTT**CTCCCCTCTCAGG**GACTTACAGTTTGAGATGC CATGACTGTGCGGTCATAAATGACTTCAACTGTCCCAACATTAGAGTATG
AB000381(3,3,4): TGTATGTGCCTCAATATTTACAAGCAGAAAATGTGAAAT CAATTATTTTCATTGCTGC**TTCTTTTTTTAGG**CATAAATTCTCGTGAACTAC TTGTTTATAAGAACTGTACAAACAACTGCACATTTGTATATGCAGCTGA
AB002059(4,1,2): GTAGGGCTCAGCTCCGCCCCTGTCACTACACGCTGGGGA CACACCACACTGCCCGACT**TCTCCTCCCCAGG**TGGGCGCTCCTCGCCAAAAA AGGCTACCAGGAGCGGGACCTGGAACCCCAGTTTTCCATCATCACCAAA
AB002059(5,2,3): AGGCTGCCGGCTTCCGGCCTTTCCAGTCAACACGAGCCC AGCCAGGCCAACCTTGAGA**CTTGCCTCCTAGG**GAGAGAACGTGTTCTTCTTG GTGACCAACTTCCTTGTGACGCCAGCCCAAGTTCAGGGCAGATGCCCAG

Table 4: 13-mers extracted from the sequences in Table 3

CCTTCCCTTC**AG**G
CTCCCCTCTC**AG**G
TTCTTTTTTT**AG**G
TCTCCTCCCC**AG**G
CTTGCCTCCT**AG**G

Total authentic 3' splice site 13-mers extracted : **2880**

3.6 CRYPTIC 3' SPLICE SITE

The cryptic 3' splice sites are collected from the DBASS3 database portal [5–7] by crawling all available splice details page. The fields on the DBASS3 splice site details pages are similar to the ones on the DBASS5 pages.
The desired 13-mer comprises of :

- 10 nucleotides before the ag dinucleotide from the Intron
- the ag dinucleotide at the splice site marker from the Intron
- one nucleotide after the splice site marker from the Exon

Consider the following nucleotide sequence extracted from a record at URL: http://www.dbass.org.uk/DBASS3/viewsplicesite.aspx?id=52 (last retrieved: 04/23/2017)

gtaagggccg ggggcatttt ttctttctta aaaaaatttt tttttaagag atgggttctt gctatgctgc ccaggctggt cttaaattcc tagtctcaaa tgatcctccc acctcagcct caagtgtgag ccacctttgg ggcatcccca atccaggtcc ctggaagctc ttgggggggc atatctggtg gggagaaagc aggggttggg gaggccgaag aaggtcaggc cctcagctgc cttcatca**g/** **t**tcccaccct cca**g/c**cccc **(a>g)(c>g)** ctcctcctgc agACAAGCTG GTGTCTAGGA ACTACCCGGA CCTGTCCTTG GGAGACTACT CCCTGCTCTG GAAAGCCCAC AAGAAGCTCA CCCGCTCAGC CCTGCTGCTG GGCATCCGTG ACTCCATGGA GCCAGTGGTG GAGCAGCTGA CCCAGGAGTT CTGTGAGgta

We can observe from other details on the page that:

- Capital nucleotide symbols indicate the leading part of the Exon
- Small nucleotide symbols indicate the trailing part of the Intron
- Mutations are marked as : (a>g) and (c>g) Cryptic splice sites are marked as : "cagctgc cttcatcag/ ttcccaccct ccag/cccc"

There are two cryptic 3' splice site 9-mers extracted from the above sequence. They are: "tgccttcatcagt" and "cccaccctccagc"

Total cryptic 3' splice site 13-mers extracted : **306**

3.7 NEIGHBORING 5' SPLICE SITE

The putative splice sites around the known cryptic splice sites are called as neighboring splice sites. These are the putative splice sites that were not selected for splicing by the spliceosome in the event of a mutation. Using the nucleotide sequences extracted from the DBASS5 splice site details pages. The 100 base-pairs upstream and 100 base-pairs downstream of the splice site are parsed to look for the GT dinucleotide. All such occurrences are captured in the neighboring 5' splice site dataset in the form of 9-mers with the GT dinucleotide as the 4th and 5th bases.

Total neighboring 5' splice site 9-mers extracted : **2213**

4 ALGORITHM SELECTION

In case of 5' splice sites, we can model the problem statement as a search problem. The spliceosome binds to a specific subset of the entire 9-mer search space. The cardinality of each position in the 9-mer is four (equivalent to A, C, G, and T), and the positions 4 and 5 are always occupied by the GT di-nucleotide. Hence, the size of the search space is $4^7 = 16384$. This search space comprises of the authentic, cryptic, and neighboring splice sites. An important goal is to study the statistical properties of the known samples from these three categories of splice sites and prove that they are inherently different.

Evolutionary Algorithms(EAs) are applicable to problems that have no known classical optimization methods [9]. If a traditional method is applicable to solve a problem, then EA should not be used since traditional methods are more efficient in such a case. Since we do not understand the choices of a spliceosome completely, EAs are suitable for this problem. The section-5 describes a Genetic Algorithm based approach.

Three primary forms of EA:

- Evolutionary Programming
- Genetic Algorithms
- Evolutionary Strategies

Some of the early significant contributors to EA:

- Friedberg(1958)
- Fraser(1957)
- Bremermann(1962)
- Box and Draper(1969): Evolutionary Operation

- Spendley(1962)
- Fogel(1966): Evolutionary Programming
- Holland(1967): Genetic Algorithms
- Rechenberg(1965): Evolutionary Strategies

Some of the initial conferences on EA:

- Schwefel and Manner(1991): Int'l workshop
- Belew and Booker(1991): ICGA'91
- Fogel and Atmar(1992): EP'92
- Manner and Manderick(1992): PPSN'92

4.1 BACKGROUND ON GENETIC ALGORITHMS

Genetic Algorithms were first proposed and analyzed by John Holland(1975). Genetic Algorithms are a type of EA that deal with search and optimization based on a fitness function. It follows the philosophy of survival of the fittest. It uses the mechanisms of natural biological selection and genetics.

Genetic algorithms follow a generic domain agnostic framework. Hence, it has the advantage of being applicable to many problems. Some of the features that initially separated GA from other evolutionary approaches are:

- Bitstring representation
- Proportional selection

- Crossover

Although, the representation and selection methods have advanced significantly, there is still a large emphasis on the crossover operation. Crossover is known to give GAs a distinctive advantage over other methods [9].

General Terminology:

- Population: A collection of individuals of size m

- Individual: A single string of size m that is part of the population

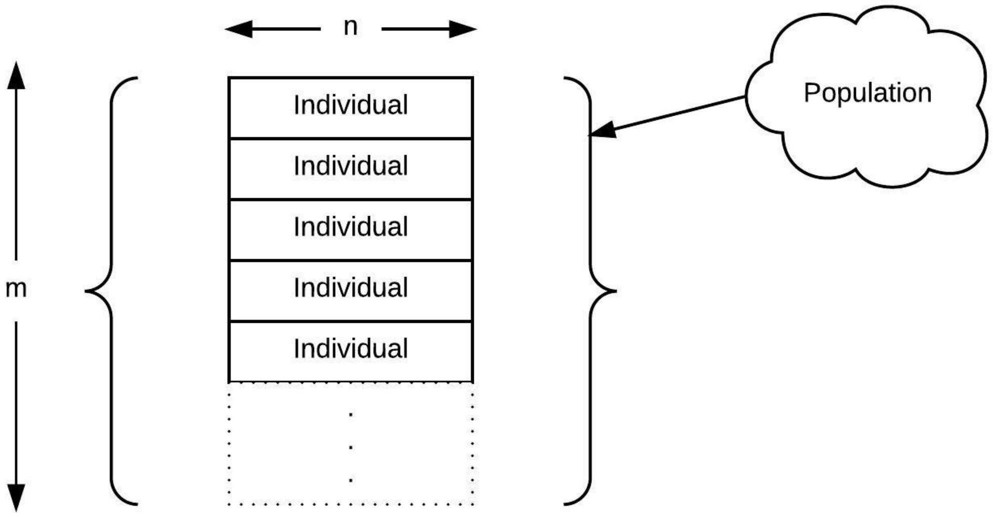


Figure 5: Population and Individuals

Biological terminology:

- Generation: equivalent to a population

- Chromosome: equivalent to an individual

- Gene: Each of the n positions in a chromosome is called a gene. It can be treated as a variable and can take values from a fixed set of alleles

- Allele: A fixed set of values that can be taken by a gene

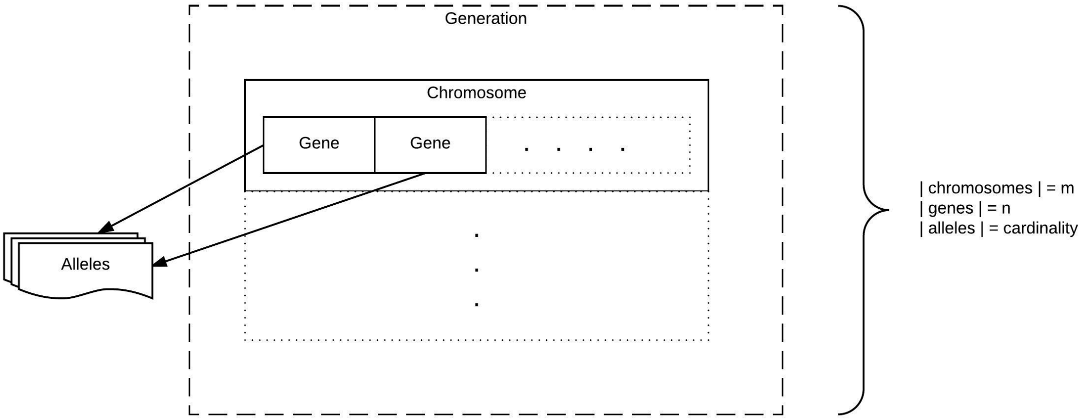


Figure 6: Chromosome, Gene, and Alleles

Figure 7 shows a generic algorithmic framework for domain agnostic genetic algorithms. The selection strategy, fitness operator, crossover operator, and recombination strategy are customizable according to a need. The framework makes no assumptions about the initial population or the operators.

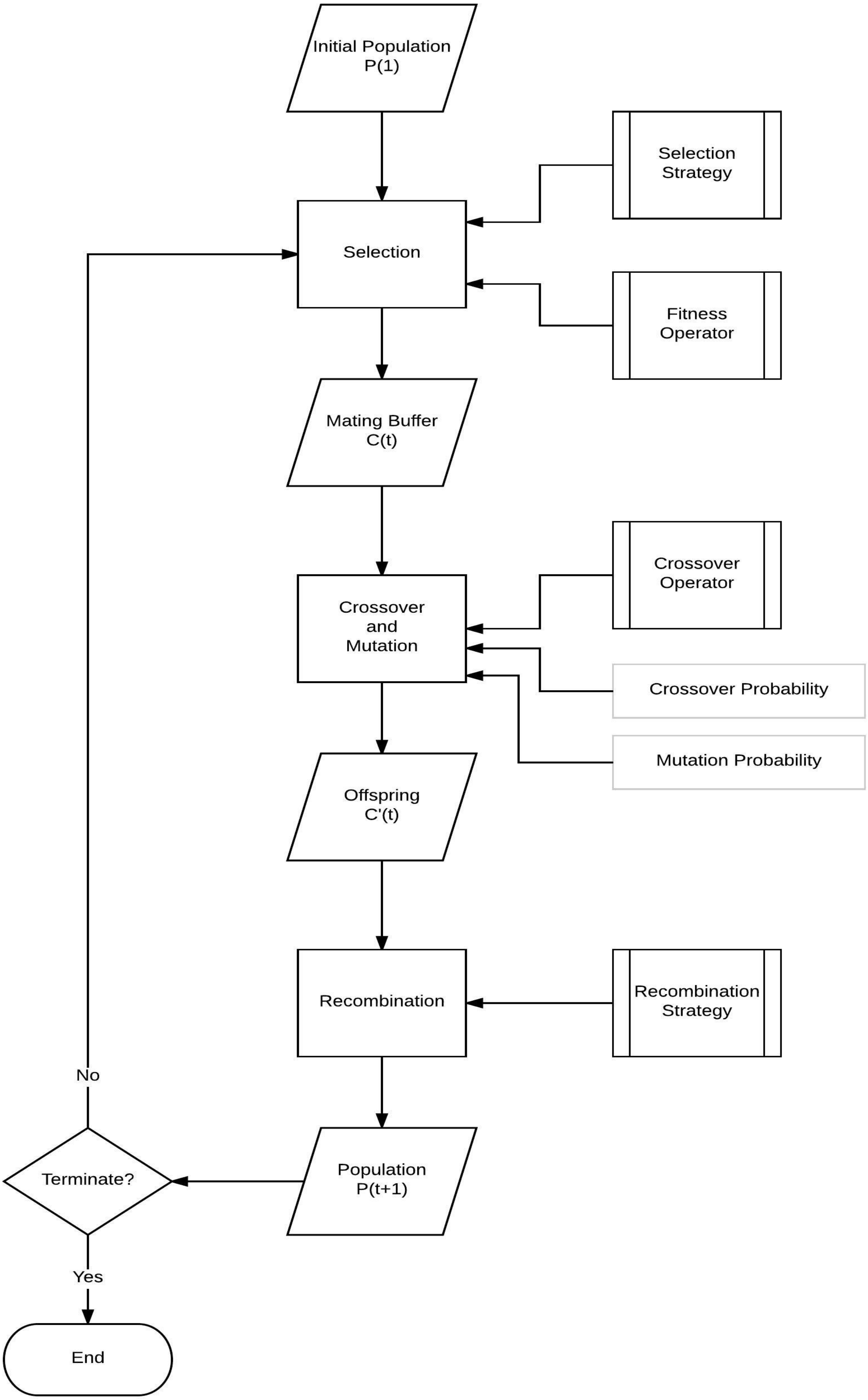


Figure 7: An algorithmic framework for genetic algorithms

The two phases that can be used to introduce bias [9]:

- Selection strategy
- Recombination / replacement strategy

4.2 SELECTION STRATEGIES

Selection strategy is responsible for selecting individuals from a population for mating.

4.2.1 ROULETTE WHEEL

Total the fitness value of all individuals in the parent population; Calculate probability of each individual as the ratio of its fitness to total fitness. This strategy may lead to weak selection pressure.

Selection pressure is governed by the differences in selection probabilities of individuals. If the differences are too small then the selection pressure is low. The workaround for low selection pressure is to scale the fitness probabilities with respect to the worst fitness. However, this leads to excessive selection pressure. The best individual may take over the entire population within a few iterations [9].

4.2.2 RANKED SELECTION

The workaround for excessive selection pressure is to use ranked selection. The parent population is sorted by fitness. Probability of selection is a linear function of the ranks sorted by fitness.

4.2.3 TOURNAMENT SELECTION

A small subset of the parent population is selected at random and the individual with best fitness is chosen for mating. This is repeated m times. The selection pressure can be controlled by adjusting the set size.

4.2.4 ELITIST STRATEGY

This is more of a strategy post selection. All but one of the child population after selection are chosen. The best individual from the parent population is added to the child population.

4.3 REPLACEMENT STRATEGIES

A child population is generated from the parent population P(t) using selection, crossover, and mutation operations. A suitable replacement strategy is required to form the population for the next generation P(t+1) using candidate individuals from both the parent and child populations [9].
Some of the notable GA implementations that implement complex replacement strategies are:

- Whitleys GENITOR
- Syswerda's steady-state GA
- Eshelman's CHC
- Mühlenbein's breeder GA

4.3.1 REPLACEMENT STRATEGY ABSTRACTIONS

Let us assume:

- μ: parent population
- λ: child population

4.3.2 $(\mu + \lambda)$ ES

The μ parents and λ children are merged and the best μ individuals are chosen to form the new parent population.

4.3.3 (μ, λ) ES

The μ parents produce λ offsprings such that $\lambda > \mu$. The best μ offsprings out of the λ offsprings are chosen to form the new parent population.

4.3.4 REPLACEMENT STRATEGY: OTHER VARIATIONS

Two other degrees of variation in a replacement strategy:

- Number of matings per iteration:

  Whether the GA produces one or two versus many (μ) offsprings in each iteration; De Jong and Sarma (1993) claim that the main difference between variations in number of allowed matings is that a strategy with fewer matings leads to higher variance in performance.

- Whether the replacement strategy is biased or not:

  In case of an unbiased replacement strategy, if all the parents are replaced by the children, then we risk losing good individuals from the parent population for good. An advantage of this strategy is that the algorithm may wander out of a local minimum.

4.3.5 MOST COMMON PRACTICAL REPLACEMENT STRATEGIES

[10]

- Delete All:

  All the child population replaces the entire current population

- Steady-state:

  Only n members of the current population are replaced by members from the child population; The quantity n and the strategy for removal are parametrized

- Steady-state-no-duplicates:

  Same as the Steady-state strategy except that the algorithm checks for duplicates while introducing chromosomes from the child population into the current population

4.4 MUTATION

Mutation is the mechanism for producing variations by randomly replacing one allele with another. A commonly used rate of mutation is one over length of the string.

4.5 CROSSOVER

Crossover combines features from two highly fit individuals. The individuals maybe highly fit due to different reasons. However, we do not know which features account for the high fitness. Hence, the features are combined at random [10].

Types of crossover:

- One-point crossover
- Two-point crossover
- K-point crossover
- Uniform crossover
- Uniform order-based crossover
- Order-based crossover
- Partially matched crossover (PMX)
- Cycle crossover (CX)

For more details on each type of crossover, refer to how they are used with 9-mer data in section [modeling]

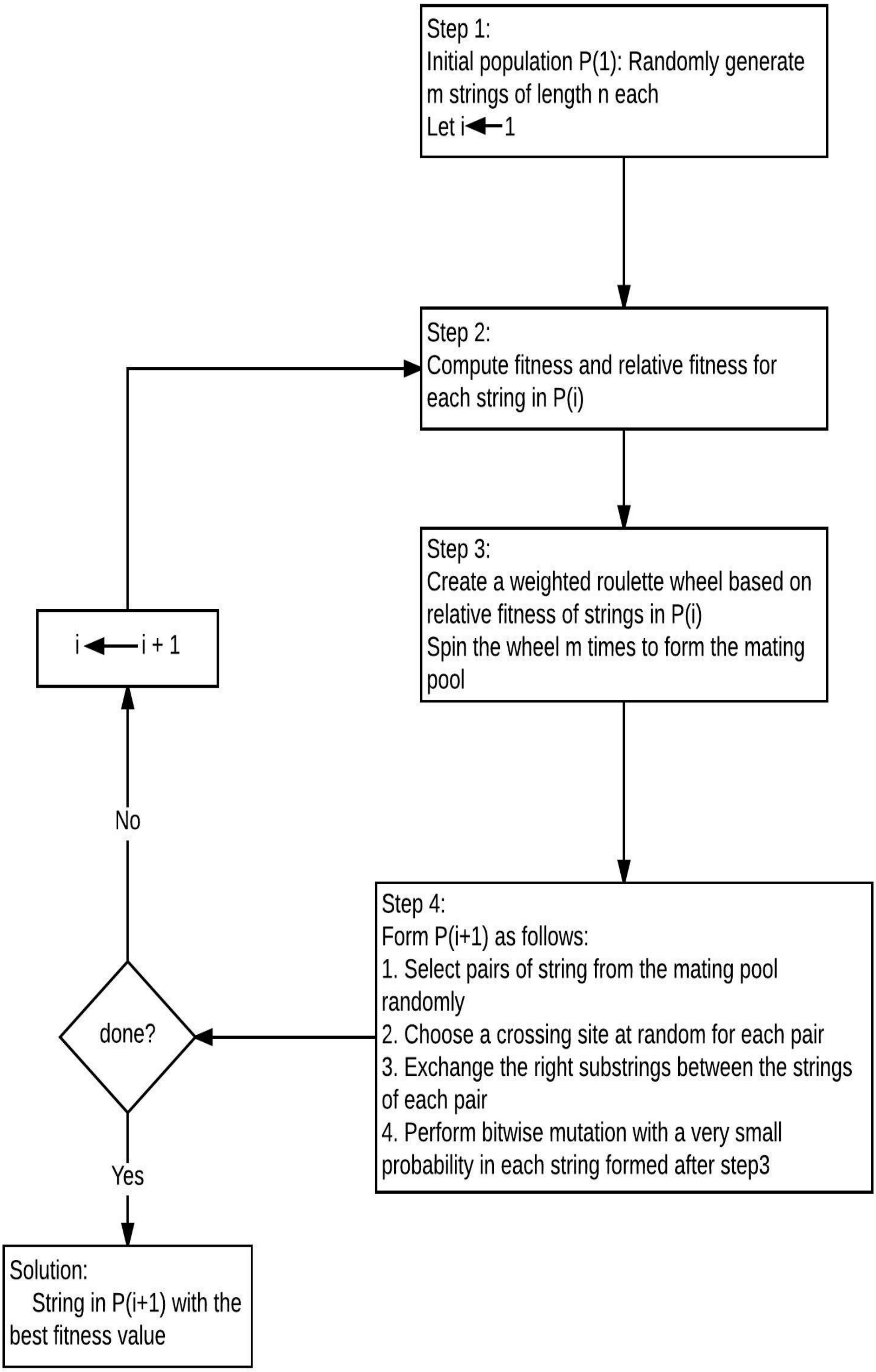

Step 1:
Initial population P(1): Randomly generate m strings of length n each
Let i ← 1
Step 2:
Compute fitness and relative fitness for each string in P(i)
Step 3:
Create a weighted roulette wheel based on relative fitness of strings in P(i)
Spin the wheel m times to form the mating pool
Step 4:
Form P(i+1) as follows:
1. Select pairs of string from the mating pool randomly
2. Choose a crossing site at random for each pair
3. Exchange the right substrings between the strings of each pair
4. Perform bitwise mutation with a very small probability in each string formed after step3
done?
No
i ← i + 1
Yes
Solution:
String in P(i+1) with the best fitness value

Figure 8: Simple GA with one-point crossover, roulette wheel selection, and delete all strategy

For more details on the crossover operators, refer how they are used with 9-mer data in the subsection-5.4.2.

5 MODELING THE PROBLEM STATEMENT USING GA

5.1 DATA MODEL

Following is a logical mapping of the terminology shown in figure 6 to the splice site n-mer data:

- chromosome: an entire n-mer
- gene: each base in the n-mer
- allele: each base in the n-mer can take one of the following four nucleotides (A: Adenine, C: Cytosine, G: Guanine, T: Thymine)

5.1.1 BASE_MAP

In order to randomize allele generation or mutation, a random integer generator is used along with a mapping of alleles to integers. Each allele is mapped to an integer and the mapping is persisted as a global constant in the code. This mapping does not change across executions.

- A: 1

- C: 2

- G: 3

- T: 4

5.1.2 9-mer

A chromosome for a 5' splice site 9-mer is represented as a string of length 9 where each character belongs to the set A, C, G, T
E.g.: AAAGTGTCC

5.1.3 13-mer

A chromosome for a 3' splice site 13-mer is represented as a string of length 13 where each character belongs to the set A, C, G, T
E.g.: AGATTCTCCCAGA

5.2 FITNESS FUNCTION: POSITION WEIGHT MATRIX

Position Weight Matrix (PWM) is a popular bioinformatic tool for predicting motifs in a sequence of nucleotides. It has also been found useful in many other applications involving sequence patterns and local multi-alignments. PWM has been successfully applied to representation of splice sites in nucleotide sequences [15]. The PWM score of a motif is a useful measure of its strength [14].

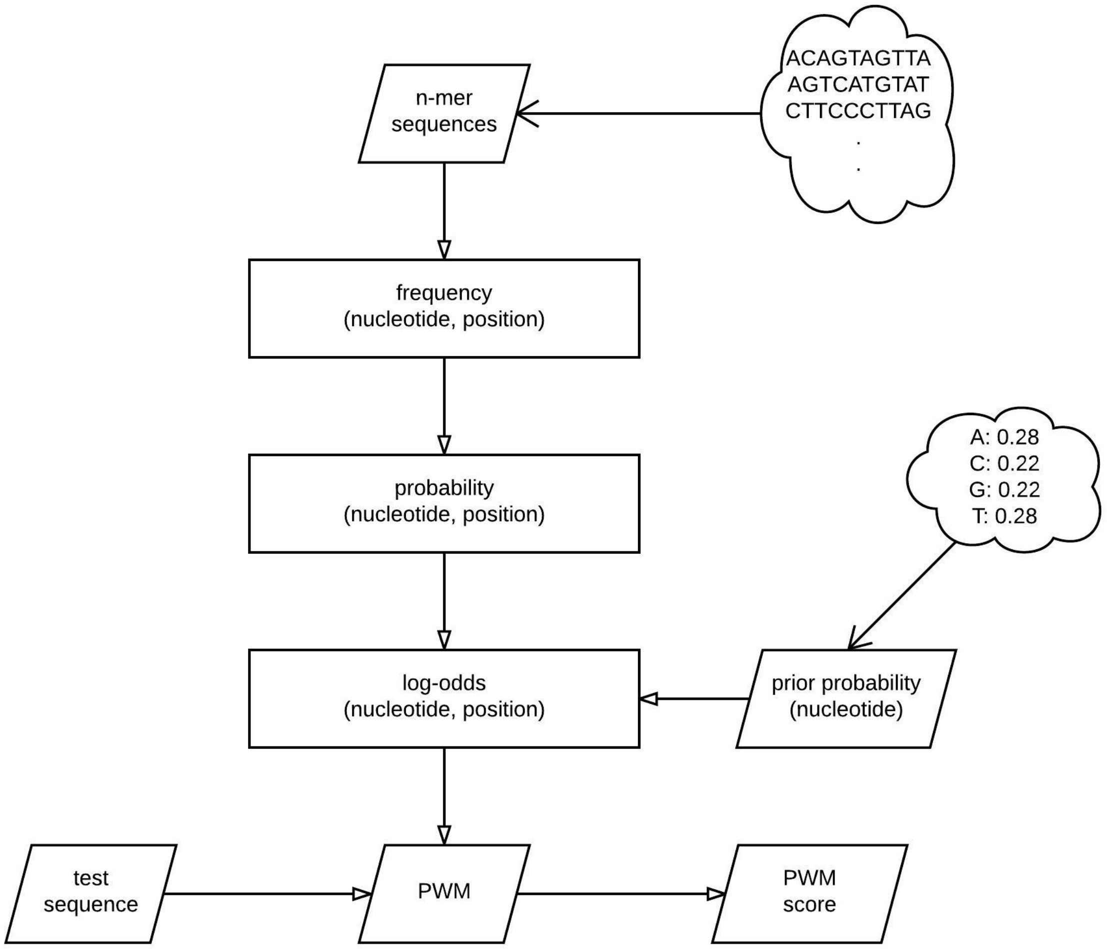

ACAGTAGTTA
AGTCATGTAT
CTTCCCTTAG
.
.
n-mer sequences
frequency (nucleotide, position)
probability (nucleotide, position)
A: 0.28
C: 0.22
G: 0.22
T: 0.28
log-odds (nucleotide, position)
prior probability (nucleotide)
test sequence
PWM
PWM score

Figure 9: Position Weight Matrix: block diagram

Diagram-5.2 shows the different phases involved in computation of a PWM from nucleotide sequences. **Algorithm: PWM** Given Inputs:

- Nucleotide sequences
- Nucleotide prior probabilities

```
        symbolOddsMap = {A:0.28, C:0.22, G:0.22, T:0.28}
        freqMat = frequencyCounts(data, symbolSet, columnCount)
        laplacePWMMat = laplaceCountPWM(freqMat, symbolSet)
        logoddsMat = logodds(laplacePWMMat, symbolSet,
           symbolOddsMap)
```

The logOdds matrix computed in step 4 is the PWM and can be used to score unknown sequences.

Laplace pseudo count is used to handle zero values in step 3.

The logodds matrix is the logarithm of a likelihood ratio. Hence,

score $> 0 \Rightarrow$ positive sample

score $< 0 \Rightarrow$ negative sample

5.3 ALGORITHMS: GA

The high-level generic flow of the algorithm is as follows:

1. Build a probabilistic model of the input dataset
2. Start with a random population
3. Extrapolate a subset of the search space using the probabilistic model built in step 1
4. Determine how much of the input dataset is covered by the subset of the search space extrapolated in step 3

The input dataset comprises of the following types of splice sites:

- A_5: All known authentic 5' splice sites
- C_5: All known cryptic 5' splice sites
- N_5: All known neighboring 5' splice sites extracted as specified in section 3.7
- A_3: All known authentic 3' splice sites

- C_3: All known cryptic 3' splice sites

For the sake of simplicity, let us consider only the 5' splice site datasets: $A_5, C_5, andN_5$. Let S_5 be the entire search space of 5' splice sites.

$$|S_5| = 4^7 = 16384$$

Without making any assumptions about the relation that exists between each dataset and S_5, we can state:

$$A_5 \subset S_5$$

$$C_5 \subset S_5$$

$$N_5 \subset S_5$$

Now, we can define the following functions that map the entire search space to boolean values indicating membership to the known set of splice sites.

$$fcrypt_5 : S_5 \rightarrow Boolean; fcrypt_5(x) = \text{True } \forall x \in C_5$$

$$fauth_5 : S_5 \rightarrow Boolean; fauth_5(x) = \text{True } \forall x \in A_5$$

$$fsplice_5 : S_5 \rightarrow Boolean; fsplice_5(x) = \text{True } \forall x \in \{A_5 U C_5\}$$

Similarly, for 3' splice sites, we can define the entire search space S_3 such that:

$$|S_3| = 4^{11} = 4194304$$

and we can define similar membership functions as follows:

$$fcrypt_3 : S_3 \rightarrow Boolean; fcrypt_3(x) = \text{True } \forall x \in C_3$$

$$fauth_3 : S_3 \rightarrow Boolean; fauth_3(x) = \text{True } \forall x \in A_3$$

$$fsplice_3 : S_3 \rightarrow Boolean; fsplice_3(x) = \text{True } \forall x \in \{A_3 U C_3\}$$

Now, we can redefine the problem statement defined in section-2 as follows: Can each membership function on chromosomes (fcrypt, fauth and fsplice) be defined in terms of simpler functions of the individual genes?

We can use Genetic Algorithms to randomly extrapolate a subset of the search space S_5 with the help of a suitable fitness function. PWM is a highly sensible measure of statistical distribution of nucleotides in an n-mer. Hence, the PWM score of an n-mer is used as the fitness function for the Genetic Algorithm. The extrapolated subset of S_5 should have a high degree of similarity to the dataset used to train the PWM as compared to other datasets.

5.3.1 COMPUTING EXACT MATCH RATIO (EMR)

Let,

- PWM_A_5 : PWM trained using A_5
- PWM_C_5 : PWM trained using C_5
- PWM_N_5 : PWM trained using N_5
- EA_5 : Subset of S_5 extrapolated using GA and PWM_A_5

- EC_5 : Subset of S_5 extrapolated using GA and PWM_C_5
- EN_5 : Subset of S_5 extrapolated using GA and PWM_N_5

We compare the EA_5, EC_5, and EN_5 subsets based on their degree of similarity to the subsets A_5, C_5, and N_5. For the sake of simplicity, we compute degree of similarity using exact match ratio (EMR).

Given two subsets S1 and S2, EMR can be defined as follows:

$$EMR(S1, S2) = |S2 - S1|/|S2|$$

For example, consider the following comparisons:

$EMR(EA_5, A_5) = |A_5 - EA_5|/|A_5|$: Percentage of chromosomes in A_5 that are covered by EA_5

This can also be written as : $EMR(EA_5, A_5) = |A_5 \cap EA_5|/|A_5|$

Similarly, we can define $EMR(EA_5, C_5) = |C_5 \cap EA_5|/|C_5|$

In simpler terms, EMP(S1, S2) = ratio of individual n-mer strings in S2 covered by S1

5.3.2 COMPETING DATASETS

Two datasets are competing if they are supposed to be intrinsically different as per the hypothesis. Thus,

- A_5 competes C_5
- C_5 competes A_5
- N_5 competes $A_5 \cup C_5$

5.3.3 COMPARISON SCORES

Using the definition of EMR, we can define the score metrics that quantify the degree of similarity for comparing the datasets.

Let us assume two datasets P and Q and their extrapolated sets EP and EQ (using GA with PWM). If P and Q are competing datasets(defined in section-5.3.2, we can define the following comparison scores:

- PRESERVATION SCORE (PSCORE)

$$PSCORE(P) = EMR(EP, P)$$

$$PSCORE(Q) = EMR(EQ, Q)$$

- CONFUSION SCORE (CSCORE)

$$CSCORE(P) = 1 - EMR(EP, Q)$$

$$CSCORE(Q) = 1 - EMR(EQ, P)$$

If there exists a GA, that can give an EP and an EQ that maximize the scores PSCORE(P), PSCORE(Q), CSCORE(P), and CSCORE(Q), then we can conclude that P and Q are intrinsically different.

5.3.4 ENHANCED SCORES

The PSCORE and CSCORE measures are based on EMR, which looks for exact match across the two participating datasets. The scores of the 3' splice sites are expected to be worse than the 5' splice sites because the 3' splice sites are longer and thus less probable to exactly match the known splice sites. Hence, the EMR based scores are too rigid for longer sequences.

The coverage logic can be enhanced by considering local alignment scores instead of exact match. This should give us better coverage scores for 3' splice sites.

5.3.5 STEPS

For 5' splice sites:

Given datasets: $A_5, C_5,$ and N_5

1. Compute PWM_A_5, PWM_C_5, and PWM_N_5 from A_5, C_5, and N_5 respectively

2. Initialize a random initial population P(1) of 9-mers with the GT dinucleotide at positions 4 and 5

3. Using PWM_A_5 and a suitable GA parameters, extrapolate EA_5 from P(1)

4. Using PWM_C_5 and a suitable GA parameters, extrapolate EC_5 from P(1)

5. Using PWM_N_5 and a suitable GA parameters, extrapolate EN_5 from P(1)

6. Compute $PSCORE(A_5)$ and $CSCORE(A_5)$ given that A_5 and C_5 are competing

7. Compute $PSCORE(C_5)$ and $CSCORE(C_5)$ given that C_5 and A_5 are competing

8. Compute $PSCORE(N_5)$ and $CSCORE(N_5)$ given that N_5 and $A_5 \cup C_5$ are competing

The steps for 3' splice sites are same as above. The given datasets for 3' splice sites are $A_3, C_3,$ and N_3.

5.3.6 DATASET LIMITATIONS

For the sake of simplicity, the experiments presented in this report are performed on limited datasets that do not include all known splice sites. The datasets are specified in detail in section 3.

5.4 ALGORITHMS: OPERATORS AND STRATEGIES

5.4.1 SELECTION STRATEGIES

1. Roulette Wheel Selection:

   Example:

Table 5: Sample data: Roulette Wheel Selection

Label	Chromosome	Fitness	Probability	Cumulative Probability
C1	1101010	30	$30/80 = 0.375$	0.375
C2	1011001	15	$15/80 = 0.1875$	0.5625
C3	0100110	10	$10/80 = 0.125$	0.6875
C4	1000101	5	$5/80 = 0.0625$	0.75
C5	1111000	20	$20/80 = 0.25$	1

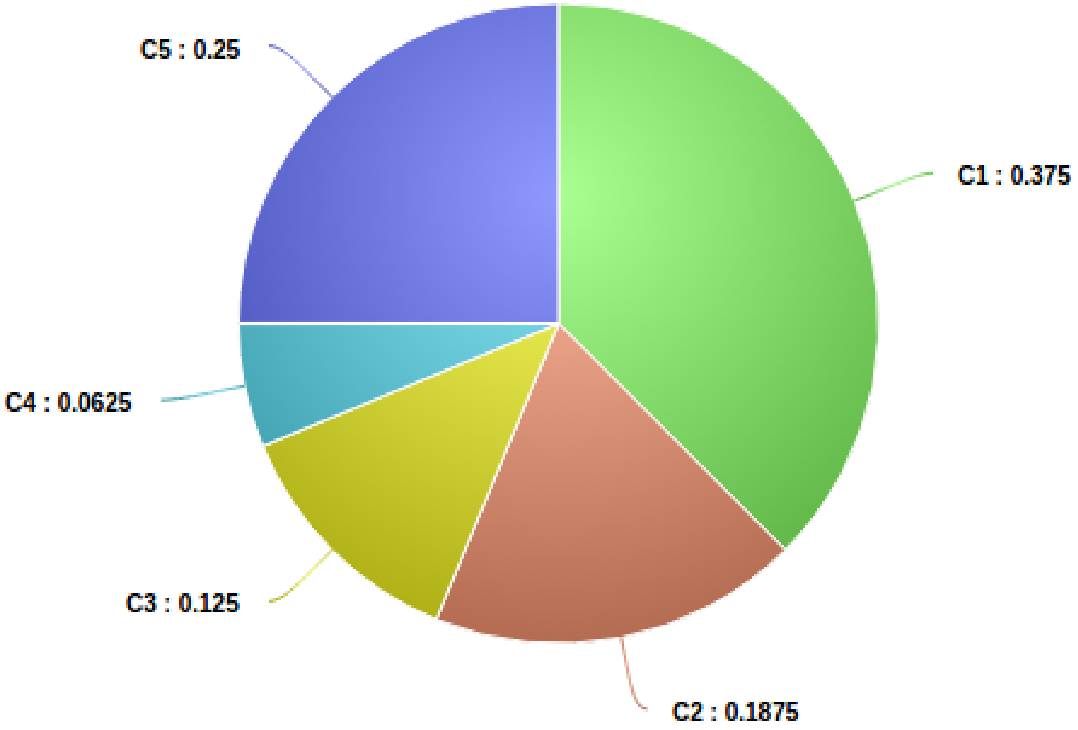


Figure 10: Roulette wheel for chromosomes in Table 5

Algorithm:

A random number generator is used to generate a value $r\epsilon[0, 1]$. The cumulative probabilities are used to select a chromosome. For example, if r $= 0.7$, then C4 is selected since $0.6875 < 0.7 < 0.75$

```python
def rouletteWheel(population, fitnessFunction):
    fitnessArr = []
    for solution in population:
        fitness = fitnessFunction(solution) # compute fitness of each chromosome
        fitnessArr.append(fitness)

    minFitness = min(fitnessArr)
    fitnessArr = numpy.divide(fitnessArr, float(minFitness)) #scaling fitness values
```

```
        popFitness = float(sum(fitnessArr))
        normFitnessArr = map(lambda x: x / popFitness, fitnessArr)  # normalize fitnessArr

        cumulative = 0
        normFitnessCumulativeArr = []
        for norm in normFitnessArr: # compute cumulative normalized fitness values
            cumulative = cumulative + norm
            normFitnessCumulativeArr.append(cumulative)

        ret = []
        for i in range(len(population)):
            spin = random.random()
            for j in range(1, len(normFitnessCumulativeArr)):
                if spin < normFitnessCumulativeArr[j]:
                    ret.append(population[j-1])
                    break
        return ret
```

2. Ranked Selection:

   Example:

   Recomputing probabilities of the sample chromosomes from Table 5:

Table 6: Sample data: Ranked Selection

Label	Chromosome	Fitness	Rank	Probability	Cumulative Probability
C1	1101010	30	5	$5/15 = 0.333$	0.333
C2	1011001	15	3	$3/15 = 0.2$	0.533
C3	0100110	10	2	$2/15 = 0.133$	0.666
C4	1000101	5	1	$1/15 = 0.067$	0.733
C5	1111000	20	4	$4/15 = 0.267$	1

Algorithm:

We can re-use the roulette wheel selection algorithm after recomputing the fitness probabilities as a linear function of the ranks.

```python
def ranked(population, fitnessFunction):
    fitnessTupleArr = []
    for solution in population:
        fitness = fitnessFunction(solution)
        fitnessTupleArr.append((solution, fitness))
    fitnessTupleArrSorted = sorted(fitnessTupleArr, key = lambda x:x[1]) # sort by fitness

    count = len(fitnessTupleArrSorted)
    popFitness = count * (count+1) / 2.0 # simulating rank order

    normFitnessTupArr = []
    idx = 1
    for tup in fitnessTupleArrSorted:
        rankProb = idx / popFitness
        normFitnessTupArr.append((tup[0], rankProb))
        idx += 1

    cumulative = 0
    normFitnessCumulativeArr = []
    for norm in normFitnessTupArr:
        cumulative = cumulative + norm[1]
        normFitnessCumulativeArr.append(cumulative)

    ret = []
    for i in range(len(population)):
        spin = random.random()
        for j in range(1, len(normFitnessCumulativeArr)
           ):
            if spin < normFitnessCumulativeArr[j]:
                ret.append(fitnessTupleArrSorted[j
                   -1][0])
                break
    return ret
```

3. Tournament Selection:

   Algorithm:

   A fixed number (s) of random chromosomes are selected for a tournament. The chromosome with the highest fitness value among the s chromosomes is added to the mating pool. This process is repeated M times to select M chromosomes.

```
def tournament(population, fitnessFunction, s=2):
    ret = []
    m = len(population)
    for idx in range(m):
        compete = random.sample(population, s)
        competeFitness = [fitnessFunction(individual)
           for individual in compete]
        winnerIdx = numpy.argmax(competeFitness)
        winner = compete[winnerIdx]
        ret.append(winner)
    return ret
```

5.4.2 CROSSOVER OPERATORS

1. One Point Crossover:

   Example:

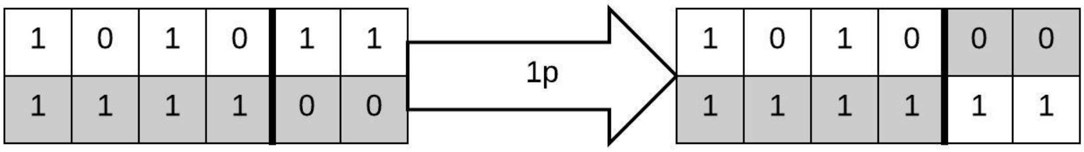


Figure 11: Example: One Point Crossover

Algorithm:

A crossover site is selected at random and all alleles on one side of the site are exchanged between the two parents.

```
def uniform(parent1, parent2, swap_prob = 0.5):
    parentLen = len(parent1)
    child1 = parent1[:]
    child2 = parent2[:]
    for idx in range(parentLen):
        spin = random.random()
        if spin > 0.5: # perform allele swap at idx
```

```
            swap(child1, child2, idx)
    return (child1, child2)
```

2. Two Point Crossover:

   Example:

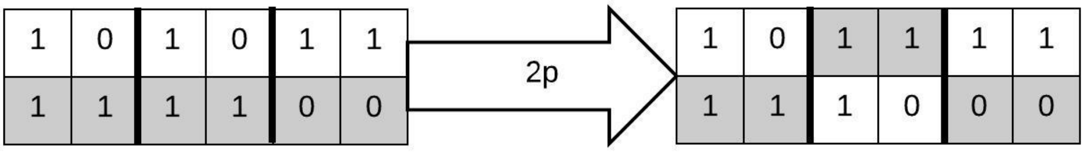


Figure 12: Example: Two Point Crossover

Algorithm:

Two crossover sites are selected at random and all alleles between the two sites are exchanged between the two parents.

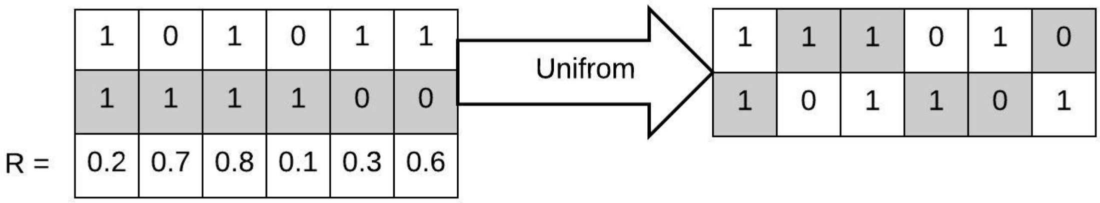
```
def two_point(parent1, parent2):
    parentLen = len(parent1)
    site1 = pick_random_site(parentLen)
    site2 = pick_random_site(parentLen, negativeSites=[
       site1])
    site1, site2 = sorted((site1, site2))
    p1_sub = parent1[site1:site2+1]
    p2_sub = parent2[site1:site2+1]
    child1 = parent1[0:site1] + p2_sub + parent1[site2
       +1:]
    child2 = parent2[0:site1] + p1_sub + parent2[site2
       +1:]
    return (child1, child2)
```

3. k-Point Crossover:

   The concept of 1-point or 2-point crossover can be extended to k points. The alleles between each odd crossover point and its immediately successive even crossover point are exchanged between the two parents.

4. Uniform Crossover:

   Example:



	1	0	1	0	1	1
	1	1	1	1	0	0
R =	0.2	0.7	0.8	0.1	0.3	0.6

1	1	1	0	1	0
1	0	1	1	0	1

Figure 13: Example: Uniform Crossover

Algorithm:

Each gene in the chromosome is considered for crossover with some swapping probability p_e. The alleles at the chosen gene locations ($R > p_e$) are swapped between the two parents.

```python
def uniform(parent1, parent2, swap_prob = 0.5):
    parentLen = len(parent1)
    child1 = parent1[:]
    child2 = parent2[:]
    for idx in range(parentLen):
        spin = random.random()
        if spin > 0.5: # perform allele swap at idx
            swap(child1, child2, idx)
    return (child1, child2)
```

5. Uniform Order-Based Crossover: Example:

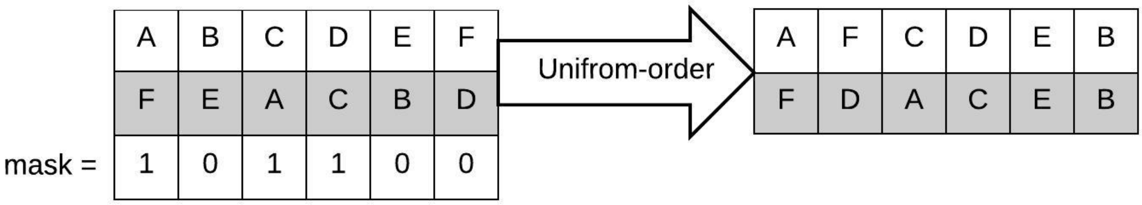


Figure 14: Example: Uniform Order-Based Crossover

Algorithm:

A random binary template is used to guide the crossover. The alleles at indices in the parents corresponding to a 0 in the mask are reordered according to their order in the other chromosome.

```
    def uniform_orderbased(parent1, parent2):
        parentLen = len(parent1)
        child1 = parent1[:]
        child2 = parent2[:]
        maskStr = bin(random.getrandbits(parentLen)).
           replace('0b','')
        prefix = '0'*(parentLen - len(maskStr))
        maskStr = prefix + maskStr
        c1mask = [int(c) for c in maskStr]
        c2mask = [int(c) for c in maskStr]

        for neg in negativeSites:
            c1mask[neg] = 1
            c2mask[neg] = 1

        c1SwapSet = [child1[idx] for idx in range(parentLen
           ) if c1mask[idx] == 0]
        c2SwapSet = [child2[idx] for idx in range(parentLen) if c2mask[idx] == 0]
        for idx in range(parentLen):
            allele = parent2[idx]
            if allele in c1SwapSet and 0 in c1mask:
                c1Idx = c1mask.index(0)
                child1[c1Idx] = allele
                c1mask[c1Idx] = 1

            allele = parent1[idx]
            if allele in c2SwapSet and 0 in c2mask:
                c2Idx = c2mask.index(0)
                child2[c2Idx] = allele
                c2mask[c2Idx] = 1

        return (child1, child2)
```

6 RESULTS

The algorithm defined in the steps section-5.3.5 is executed for each combination of the following parameters for GA:

- generation count: {10, 100, 1000} - number of generations
- generation size: {10, 20} - size of each generation
- selection strategy: {roulette wheel, ranked, tournament}
- crossover operator: {one point, two point, uniform, order based, partially matched, cycle}

6.1 SCORING TABLES

The scoring tables are computed by executing each combination of the GA parameters for 100 iterations. For the set of parameter values defined above, there are 3 X 2 X 3 X 6 = 108 combinations. Each combination is executed 100 times in order to account for the stochastic nature of the results of each execution.

Format of the scoring table:

- title of the table indicates the selection strategy and the crossover operator
- rows 1 - 3 indicate the dataset(AUTH, CSS, or NEIGHBOR), the score metric(PSCORE, CSCORE or FITNESS), and the GA paramters m(generation size) and gc(generation count)
- for a single set of parameter values for m and gc, there is a composite column comprising of two subcolumns f($\bar{x}$) and N.
  f($\bar{x}$) = unique value of the score metric
  N = frequency of the f($\bar{x}$) value across 100 execution
- the last row contains the mean of the score metric across all 100 execution
- row 5 to second last row contain the top eight f($\bar{x}$) values based on their frequency across the 100 executions

The scoring tables for the best performing parameter combination for each dataset with respect to the PSCORE, CSCORE, and FITNESS are listed in this section

6.2 SCORING TABLES: 5’

Table 7: SELECTION: ROULETTEWHEEL; CROSSOVER: 2P: AUTH - CSCORE

AUTH - compete											
AUTH-10		AUTH-10		AUTH-10		AUTH-20		AUTH-20		AUTH-20	
m = 10, gc = 10		m = 10, gc = 100		m = 10, gc = 1000		m = 20, gc = 10		m = 20, gc = 100		m = 20, gc = 1000	
f($\bar{x}$)	N	f($\bar{x}$)	N	f($\bar{x}$)	N	f($\bar{x}$)	N	f($\bar{x}$)	N	f($\bar{x}$)	N
1.0	100	1.0	84	1.0	42	1.0	97	1.0	89	1.0	63
1.0	100	1.0	84	1.0	42	1.0	97	1.0	89	1.0	63
		0.4	2	1.0	42	0.95	3	0.9	1	1.0	63
		0.8	1	0.1	5			0.8	1	0.55	2
		0.6	1	0.4	8			0.95	9	0.8	3
		0.9	11	0.2	1					0.4	1
		0.7	1	0.3	1					0.65	1
				0.6	5					0.85	2
				0.8	6					0.1	1
f̂ = 1.0		f̂ = 0.968		f̂ = 0.779		f̂ = 0.9985		f̂ = 0.9925		f̂ = 0.925	

Table 8: SELECTION: ROULETTEWHEEL; CROSSOVER: UNIFORMORDER-BASED: CSS - FITNESS

CSS - fitness											
CSS-10		CSS-10		CSS-10		CSS-20		CSS-20		CSS-20	
m = 10, gc = 10		m = 10, gc = 100		m = 10, gc = 1000		m = 20, gc = 10		m = 20, gc = 100		m = 20, gc = 1000	
f($\bar{x}$)	N	f($\bar{x}$)	N	f($\bar{x}$)	N	f($\bar{x}$)	N	f($\bar{x}$)	N	f($\bar{x}$)	N
10.2745848678	7	9.6636271585	17	10.5546927869	28	7.86063569485	7	9.79887986855	8	9.79887986855	24
10.2745848678	7	9.6636271585	17	10.5546927869	28	7.86063569485	7	9.79887986855	8	9.79887986855	24
9.52939876679	2	10.2745848678	15	10.2745848678	13	7.47156717343	1	9.32501268826	1	9.25645354096	1
7.09569742324	1	9.67078985187	2	9.83010359059	1	8.57170792736	1	6.83397962429	1	8.27497200576	1
6.74569466089	1	9.17550294324	3	10.4204643952	3	6.31901440884	1	8.27497200576	1	9.6636271585	1
7.54206517862	1	7.67762193286	1	9.79887986855	4	10.2745848678	3	9.93310826026	1	8.15737161645	1
7.03503517961	1	7.76382598768	2	7.97163643169	1	5.62964325793	1	7.85805135572	1	10.2745848678	4
10.2745848678	7	8.31388111948	1	7.73996605529	1	8.79061043701	1	9.67078985187	1	6.12665944794	1
6.49190240736	1	6.60836516393	1	8.69979794403	1	9.6789085118	1	3.07984021931	1	9.17550294324	1
f̂ = 8.17125686486		f̂ = 8.90459883956		f̂ = 9.83882318289		f̂ = 7.85153157327		f̂ = 8.47527857503		f̂ = 9.03385109483	

Table 9: SELECTION: ROULETTEWHEEL; CROSSOVER: UNIFORMORDER-BASED: NEIGHBOR - pscore

NEIGHBOR - self											
NEIGHBOR-10		NEIGHBOR-10		NEIGHBOR-10		NEIGHBOR-20		NEIGHBOR-20		NEIGHBOR-20	
m = 10, gc = 10		m = 10, gc = 100		m = 10, gc = 1000		m = 20, gc = 10		m = 20, gc = 100		m = 20, gc = 1000	
$f(\bar{x})$	N	$f(\bar{x})$	N	$f(\bar{x})$	N	$f(\bar{x})$	N	$f(\bar{x})$	N	$f(\bar{x})$	N
0.0	38	0.0	18	1.0	18	0.1	21	0.1	16	0.15	15
0.0	38	0.0	18	1.0	18	0.1	21	0.1	16	0.15	15
0.5	4	0.5	5	1.0	18	0.0	6	0.5	2	0.25	2
0.2	10	0.1	18	0.1	13	0.1	21	0.85	3	0.3	4
0.8	1	0.2	16	0.2	9	0.05	20	0.05	13	0.8	9
0.4	7	1.0	17	0.4	4	0.3	3	0.2	12	0.1	10
0.3	9	0.8	1	0.6	10	0.15	18	0.8	2	1.0	2
0.1	31	0.6	6	0.3	4	0.2	14	0.3	7	0.05	7
		0.3	7	0.8	7	0.35	4	0.15	10	0.65	1
$\hat{f} = 0.134$		$\hat{f} = 0.385$		$\hat{f} = 0.55$		$\hat{f} = 0.144$		$\hat{f} = 0.2905$		$\hat{f} = 0.451$	

6.3 SCORING TABLES: 3’

Table 10: SELECTION: ROULETTEWHEEL; CROSSOVER: 2P: CSS - CSCORE

CSS - compete											
CSS-10		CSS-10		CSS-10		CSS-20		CSS-20		CSS-20	
m = 10, gc = 10		m = 10, gc = 100		m = 10, gc = 1000		m = 20, gc = 10		m = 20, gc = 100		m = 20, gc = 1000	
$f(\bar{x})$	N	$f(\bar{x})$	N	$f(\bar{x})$	N	$f(\bar{x})$	N	$f(\bar{x})$	N	$f(\bar{x})$	N
1.0	100	1.0	97	1.0	89	1.0	100	1.0	94	1.0	81
1.0	100	1.0	97	1.0	89	1.0	100	1.0	94	1.0	81
		0.9	2	0.9	8			0.9	1	0.9	2
		0.1	1	0.8	3			0.85	2	0.8	1
								0.95	3	0.95	16
$\hat{f} = 1.0$		$\hat{f} = 0.989$		$\hat{f} = 0.986$		$\hat{f} = 1.0$		$\hat{f} = 0.9945$		$\hat{f} = 0.988$	

Table 11: SELECTION: ROULETTEWHEEL; CROSSOVER: UNIFORMORDER-BASED: AUTH - FITNESS

AUTH - fitness											
AUTH-10		AUTH-10		AUTH-10		AUTH-20		AUTH-20		AUTH-20	
m = 10, gc = 10		m = 10, gc = 100		m = 10, gc = 1000		m = 20, gc = 10		m = 20, gc = 100		m = 20, gc = 1000	
f($\bar{x}$)	N	f($\bar{x}$)	N	f($\bar{x}$)	N	f($\bar{x}$)	N	f($\bar{x}$)	N	f($\bar{x}$)	N
6.3310505573	7	14.4612478686	48	14.4612478686	51	5.60441620002	11	5.60441620002	7	11.6082184986	5
6.3310505573	7	14.4612478686	48	14.4612478686	51	5.60441620002	11	5.60441620002	7	11.6082184986	5
12.7507896657	1	10.9189661574	1	7.12744429663	1	-0.312763542694	1	3.6195612137	1	2.78157522191	1
8.74171451452	1	13.5338392434	1	8.52921778081	1	4.35820017723	1	2.18544963501	1	3.74587732303	1
10.8756868439	1	10.3421469139	1	12.2253609616	1	5.60003674027	1	1.71773340113	1	3.18190872128	1
12.175587014	1	11.2571484401	1	7.50321011761	1	1.89270261223	1	-0.433157556824	1	2.21196853795	1
10.3886531907	1	10.6611347935	1	12.822616703	1	6.44238486089	1	2.60414034248	1	2.9424802076	1
13.9105076821	2	9.99674996565	1	9.2834903204	1	4.78738801701	1	7.97827352522	1	1.77085644407	1
7.03896738342	1	6.99678760007	1	12.47465425	1	1.60136799488	1	4.87697425919	1	3.72888579633	1
$\hat{f}$ = 9.84240719451		$\hat{f}$ = 12.4655497918		$\hat{f}$ = 13.1848227782		$\hat{f}$ = 3.67295265604		$\hat{f}$ = 4.30809631847		$\hat{f}$ = 4.53069290039	

Table 12: SELECTION: TOURNAMENT; CROSSOVER: 1P: CSS - FITNESS

CSS - fitness											
CSS-10		CSS-10		CSS-10		CSS-20		CSS-20		CSS-20	
m = 10, gc = 10		m = 10, gc = 100		m = 10, gc = 1000		m = 20, gc = 10		m = 20, gc = 100		m = 20, gc = 1000	
f($\bar{x}$)	N	f($\bar{x}$)	N	f($\bar{x}$)	N	f($\bar{x}$)	N	f($\bar{x}$)	N	f($\bar{x}$)	N
8.40920437851	3	11.3934389795	52	11.3934389795	100	9.83636160832	3	11.3934389795	95	11.3934389795	100
8.40920437851	3	11.3934389795	52	11.3934389795	100	9.83636160832	3	11.3934389795	95	11.3934389795	100
7.22482312359	1	10.2261364545	1			8.7530404488	1	11.3180415309	1		
10.1326487278	1	11.1190636309	3			9.61685967267	1	10.89817396	2		
9.75505859319	1	11.00926658	2			10.317592886	1	11.3934389795	95		
6.84611527489	1	11.1142743065	1			10.2568781762	1	11.1142743065	1		
9.39968525485	1	10.4433682448	1			10.2290003896	1				
8.93927422397	1	10.8310597641	1			9.53586991017	1				
7.14523144658	1	11.0175498327	1			9.262226443	1				
$\hat{f}$ = 8.43758297674		$\hat{f}$ = 11.1936370172		$\hat{f}$ = 11.3934389795		$\hat{f}$ = 9.66219155445		$\hat{f}$ = 11.3785629414		$\hat{f}$ = 11.3934389795	

Table 13: SELECTION: TOURNAMENT; CROSSOVER: 1P: CSS - PSCORE

CSS - self											
CSS-10		CSS-10		CSS-10		CSS-20		CSS-20		CSS-20	
m = 10, gc = 10		m = 10, gc = 100		m = 10, gc = 1000		m = 20, gc = 10		m = 20, gc = 100		m = 20, gc = 1000	
f($\bar{x}$)	N	f($\bar{x}$)	N	f($\bar{x}$)	N	f($\bar{x}$)	N	f($\bar{x}$)	N	f($\bar{x}$)	N
0.0	100	0.0	62	0.1	91	0.0	99	0.05	35	0.1	55
0.0	100	0.0	62	0.1	91	0.0	99	0.05	35	0.1	55
		0.7	1	0.3	1	0.05	1	0.25	2	0.45	1
		0.9	5	0.2	1			0.6	1	0.85	1
		0.8	1	0.1	91			0.05	35	0.1	55
		0.1	31					0.1	10	0.2	1
								0.4	1	0.8	1
								0.85	3	0.9	22
								0.3	2	0.15	7
$\hat{f} = 0.0$		$\hat{f} = 0.091$		$\hat{f} = 0.159$		$\hat{f} = 0.0005$		$\hat{f} = 0.3715$		$\hat{f} = 0.352$	

6.4 SCORE SUMMARY: 5'

Table 14: SUMMARY

select	xover	dataname	metric	M	N	score
ROULETTEWHEEL	2P	AUTH	cscore	10	10	1.0
ROULETTEWHEEL	1P	AUTH	fitness	10	1000	9.64759580424
TOURNAMENT	1P	AUTH	pscore	10	100	1.0
RANKED	1P	CSS	cscore	10	1000	0.984
ROULETTEWHEEL	UNIFORMORDERBASED	CSS	fitness	10	1000	9.83882318289
TOURNAMENT	1P	CSS	pscore	10	1000	1.0
TOURNAMENT	1P	NEIGHBOR	cscore	10	1000	0.995
TOURNAMENT	1P	NEIGHBOR	fitness	10	1000	6.36764544744
ROULETTEWHEEL	UNIFORMORDERBASED	NEIGHBOR	pscore	10	1000	0.55

6.5 SCORE SUMMARY: 3'

Table 15: SUMMARY

select	xover	dataname	metric	M	N	score
ROULETTEWHEEL	2P	AUTH	cscore	10	10	1.0
ROULETTEWHEEL	UNIFORMORDERBASED	AUTH	fitness	10	10	9.84240719451
TOURNAMENT	1P	AUTH	pscore	20	1000	0.3725
ROULETTEWHEEL	2P	CSS	cscore	10	10	1.0
TOURNAMENT	1P	CSS	fitness	20	10	9.66219155445
TOURNAMENT	1P	CSS	pscore	20	100	0.3715

7 ANALYSIS

7.1 5' SPLICE SITE

The 5' splice sites are represented by a 9-mer with a fixed GT di-nucleotide in positions 4 and 5. Each position in the 9-mer has a cardinality of four corresponding to the four nucleotides - Adenine, Cytosine, Guanine, and Thymine.

Search space size = $4^{9-2} = 4^7 = \mathbf{16384}$

The PWM-based genetic algorithm described in section5 extrapolates a subset of this search space. From the results described in section-6.4 we can see that the algorithm converges decently close to the known splice site data in ??? generations

Given,

Generation Count = 100

Generation size = 20

Number of 9-mers covered = $20 * 100 = 2000$

Search space coverage = 2000/16384 = 0.122

Thus, by exploring only 12.2% of the search space, we are able to approximately converge to a subset of the search space that is similar to known splice sites.

7.2 3' SPLICE SITE

The 3' splice sites are represented by a 13-mer with a fixed AG di-nucleotide in positions 11 and 12. Search space size $= 4^{13-2} = 4^{1}1 =$ **4194304**

Given,

Generation Count = 100

Generation size = 20

Number of 9-mers covered $= 20 * 100 = 2000$

Search space coverage = 2000/4194304 = 0.00047

Thus, by exploring only 0.047% of the search space, we are able to approximately converge to a subset of the search space that is similar to known splice sites. From the results described in section-6.5 we can see that the algorithm converges decently close to the known splice site data in ??? generations

8 PHASE 2: CLASSIFICATION TEST USING EXTRAPOLATED SETS

In this section, we explore the performance of a classifier trained with the splice sites extrapolated by the GA. The known splice sites are tested with this classifier to measure the accuracy of the model. The main goal is to workaround the limitations of the exact match based scoring logic discussed in the section-5.3.4. The classifier based scoring should perform better than the exact match based scores.

8.1 CLASSIFICATION PRIMER

Classification can be defined as : "Classification is the task of learning a target function f that maps each attribute set x to one of the predefined class labels y." [16]

A Classification model is a set of parameters that are learnt using a machine learning technique. The model is the target function that maps any known or unknown attribute set in the input domain space to a class label. The model is also called a classifier.

The task of Machine Learning is defined by Mitchell as : "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E."

For example, in the problem of cryptic splice site prediction:

- Task T: A binary decision of classifying an n-mer as a cryptic splice site or not
- Performance P: number of n-mers correctly classified as cryptic splice site / total number of actual cryptic splice sites
- Experience E: Parsing known cryptic splice site sequences and known authentic splice site sequences

Typically, the classifier is built using machine learning algorithms. The algorithm optimizes a function (minimize Information Gain in Decision Trees), performs a constrained optimization (maximize margin in SVM), or performs other optimizations based on input data. The classifier should fit the input data really well and be generic enough to correctly classify unknown data. The problem of fitting input data too well and performing poorly on unknown data is called overfitting. Overfitting is avoided by introducing regularization parameters into the learning process. The opposite of overfitting is underfitting, where the model is so general that it performs poorly on both input data and unknown data. Our main goal is to build a model that performs well on unknown data.

Any training algorithm is generally regulated using training parameters such as regularization, learning rate, thresholds etc. The model learnt during the training phase changes with these parameters. Apart from training itself, it is useful to find the training parameters that generate the best model. This can be achieved by splitting the data into three parts:

- Training data: 60%; Used in the training phase for current set
- Cross Validation data: 20%; Used for checking the accuracy for current set of parameters
- Test data: 20%; Used for checking the accuracy for final set of parameters

It is a difficult task to predict apriori whether an algorithm will perform well on a problem because the performance depends heavily on multiple factors:

- how the input data is modelled
- the machine learning algorithm used
- algorithm parameters
- quality of the data

Modeling the input data requires a healthy mix of domain knowledge and statistical understanding of the data. It is essential to evaluate the quality of a classifier on the given problem in order to compare the performance of different models. A popular tool for evaluating the performance of a classifier is the Receiver Operator Characteristic (R.O.C.) curves [17].

8.2 RANDOM FOREST CLASSIFIERS

Random Forest(RF) was introduced by Brieman as a bagging-based approach to decision tree classifiers [18, 19]. Bagging and boosting gained popularity as ensemble-based approaches to machine learning. Both rely on multiple classifiers and weighted voting.

AdaBoost aka Adaptive Boosting is a popular boosting technique. The basic idea is that multiple weak classifiers can be combined together to form a strong classifier. In both approaches, the training records are partitioned into n_{trees} number of bootstrap sets. In boosting, each successive tree is trained on a combination of its own bootstrap set and the misclassified records from its previous tree's bootstrap set. Whereas, in bagging, each bootstrap set is independent. In both approaches, a majority vote is taken across all weak classifiers to classify an unknown record. There are two essential algorithms related to RFs: RF training and error rate estimation.

RF training and testing approach:

1. Create n_{tree} sets of randomly selected bootstrap samples.

2. For each bootstrap sample set, randomly draw m predictors out of set of all predictors and train an unpruned decision tree. This generates an ensemble of n_{tree} decision trees.

3. Use each of the n_{tree} trees to make n_{tree} predictions for a test record. Aggregate the result by taking a majority vote.

Error Rate estimation:

1. Create an Out Of Bag (OOB) samples set i.e. the samples not in bootstrap, for each of the n_{tree} bootstrap sets.

2. Use the OOB data to make predictions by majority vote. Aggregate the results to compute OOB error rate of the Random Forest.

Random Forest parameters:

For a high number of predictors, one should generate as many trees as computationally feasible. This is required to ensure that the selection of each predictor is equiprobable. Ideally, this approach should lead to similar results across multiple executions of random forest induction.

Other parameters include number of candidate predictors selected in each tree, size of each tree, and resampling scheme or the bootstrap sample selection scheme for each iteration [18].

Most of the parameters can be estimated based on OOB frequency . The number of trees should be as large as possible. However, the optimal values of the parameters are dependent on the data itself. One should generate a random forests for all combinations of parameters and select the parameters that give the RF with minimum OOB error frequency [18].

8.3 MODELING THE RANDOM FOREST CLASSIFIER

In terms of the notations defined in section-5.3, we form two tables as follows:

$$RFtrain_5 = EA_5 \cup EC_5 \cup EN_5$$

$$RFtest_5 = A_5 \cup C_5 \cup N_5$$

The GA setting with 1000 generation that reports the maximum mean fitness is chosen. The best chromosome from each of the 1000 generations is added to the training data. This process is repeated for each authentic, cryptic, and neighboring data.

A new column is added to the tables to indicate the labels as follows: For $RFtrain_5$:

$$\text{label(x)} = 0 \; \forall x \in \{EA_5\}$$

$$\text{label(x)} = 1 \; \forall x \in \{EC_5\}$$

$$\text{label(x)} = 2 \; \forall x \in \{EN_5\}$$

For $RFtest_5$:

$$\text{label(x)} = 0 \; \forall x \in \{A_5\}$$

$$\text{label(x)} = 1 \; \forall x \in \{C_5\}$$

$$\text{label(x)} = 2 \; \forall x \in \{N_5\}$$

As shown in figure-15, the $RFtrain_5$ is used as training data to train a Random Forest classifier. The accuracy of the model is tested with the $RFtest_5$ data.

A high accuracy indicates that the training data closely captures the structure of the actual known splice sites. The training data is extrapolated from GA from a random initial population and PWM models of the known splice sites.

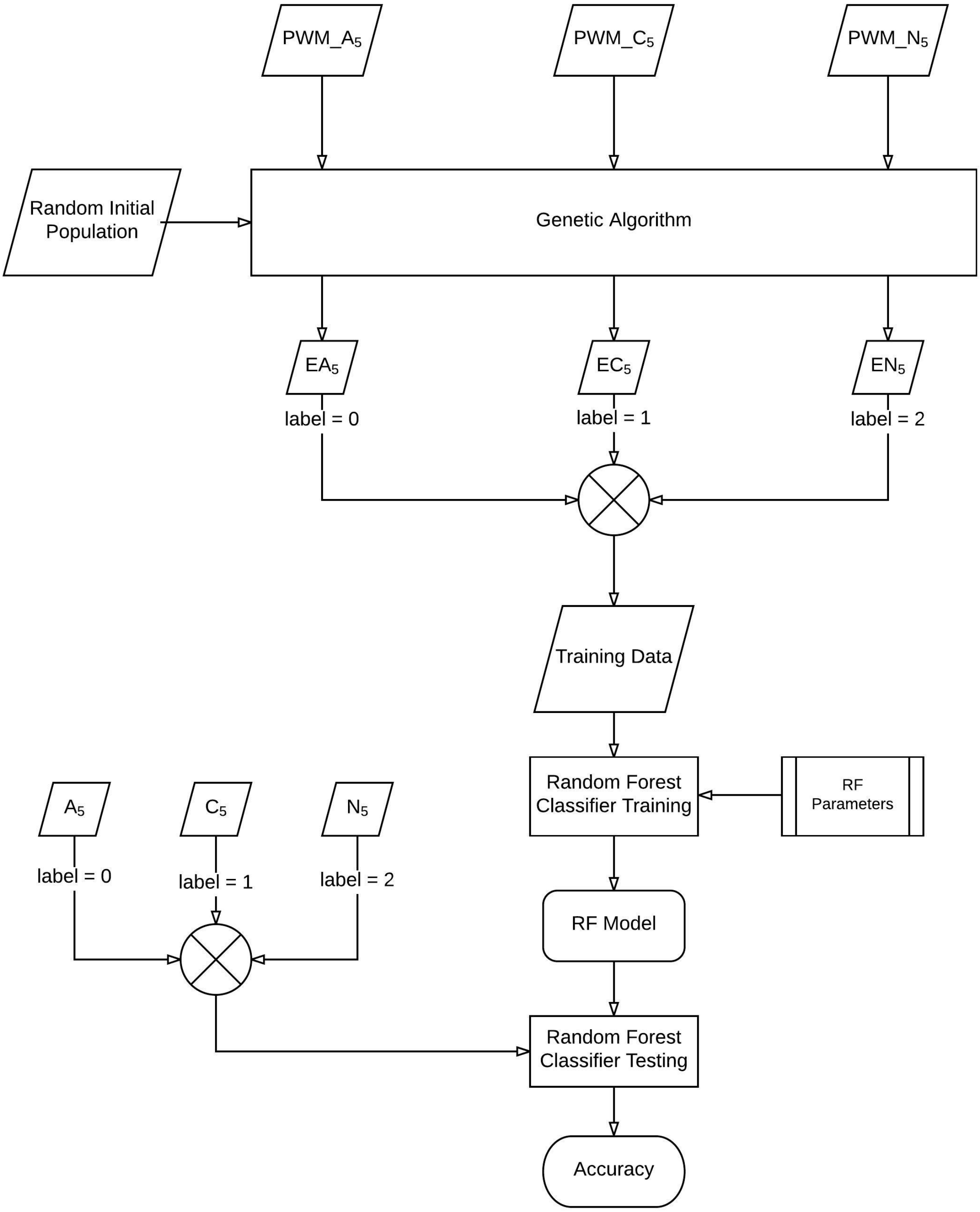


Figure 15: Random Forest Classifier workflow for extrapolated splice sites

9 CONCLUSION

As seen in the scoring summary tables, the PWM score based fitness of the n-mer population extrapolated by GA increases with the generation count. PWM score is known to be a good at modeling splice sites in nucleotide sequences [15]. This indicates that the GA is successfully traversing the search space in a controlled fashion.

Across all the generations of the splice sites, the populations with best mean scores are listed in the score summary tables in sections 6.4 and 6.5. The high preservation scores(pscore) indicate high similarity between the extrapolated set and the corresponding training splice site data. The high confusion scores (cscore) indicate high dissimilarity between the extrapolated set and the corresponding competing (section-5.3.2) splice site data.

The coverage scores for 3' splice sites are expected to be worse than 5' splice sites as explained in section-5.3.4. This can be improved by using local alignment-based logic for computing the scores.

References

[1] Crick, F. H. (1968). *The origin of the genetic code. Journal of molecular biology, 38(3), 367-379.* Available [Online]: `https://profiles.nlm.nih.gov/ps/access/scbccb.pdf` Last retrieved: 04/26/2017

[2] Wikipedia *Amino Acid* Available [Online]: `https://en.wikipedia.org/wiki/Amino_acid` Last retrieved: 04/26/2017

[3] Pollastro, P., & Rampone, S. (2003). *HS3D: homosapiens splice site data set. Nucleic Acids Research, (Annual Database).* Available [Online]: `http://www.sci.unisannio.it/docenti/rampone/` Last retrieved: 04/26/2017

[4] Pollastro, P., & Rampone, S. (2002). *HS3D, a dataset of Homo Sapiens splice regions, and its extraction procedure from a major public database. International Journal of Modern Physics C, 13(08), 1105-1117.* Available [Online]: `https://www.researchgate.net/profile/Salvatore_Rampone/publication/2549884_HS3D_a_Dataset_of_Homo_Sapiens_Splice_Regions_and_its_Extraction_Procedure_from_a_Major_Public_Database/links/0912f510aade936928000000.pdf` Last retrieved: 04/26/2017

[5] *DBASS: Database of Aberrant splice sites in human disease genes* Available [Online]: `http://www.dbass.org.uk/` Last retrieved: 04/26/2017

[6] Vorechovsky, I. (2006). *Aberrant 3' splice sites in human disease genes: mutation pattern, nucleotide structure and comparison of computational tools that predict their utilization. Nucleic acids research, 34(16), 4630-4641.* Available [Online]: `https://academic.oup.com/nar/article/34/16/4630/3111901/Aberrant-3-splice-sites-in-human-disease-genes` Last retrieved: 04/26/2017

[7] Kralovicova, J., Christensen, M. B., & Vorechovsky, I. (2005). *Biased exon/intron distribution of cryptic and de novo 3' splice*

sites. Nucleic acids research, 33(15), 4882-4898. Available [Online]: `https://academic.oup.com/nar/article/33/15/4882/2401079/Biased-exon-intron-distribution-of-cryptic-and-de` Last retrieved: 04/26/2017

[8] Buratti, E., Chivers, M., Kralovicova, J., Romano, M., Baralle, M., Krainer, A. R., & Vorechovsky, I. (2007). *Aberrant 5' splice sites in human disease genes: mutation pattern, nucleotide structure and comparison of computational tools that predict their utilization. Nucleic acids research, 35(13), 4250-4263.* Available [Online]: `https://academic.oup.com/nar/article/35/13/4250/1203203/Aberrant-5-splice-sites-in-human-disease-genes` Last retrieved: 04/26/2017

[9] De Jong, K., Fogel, D., & Schwefel, H. P. (1997). *Handbook of Evolutionary Computation. IOP Publishing Ltd and Oxford University Press.* Available [Online]: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.375.6494&rep=rep1&type=pdf` Last retrieved: 04/26/2017

[10] Sastry, K., Goldberg, D. E., & Kendall, G. (2014). *Genetic algorithms. In Search methodologies (pp. 93-117). Springer US.* Available [Online]: `http://www.graham-kendall.com/papers/sgk2014.pdf` Last retrieved: 04/26/2017

[11] Back, T., & Khuri, S. (1994, June). *An evolutionary heuristic for the maximum independent set problem. In Evolutionary Com-*

putation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on (pp. 531-535). IEEE. Available [Online]: `https://pdfs.semanticscholar.org/6881/7937ebf74d20c68e04ecc9b4aa04a4e8d8dc.pdf` Last retrieved: 04/26/2017

[12] Khuri, S. (2012). *Lecture Notes on Genetic Algorithms*

[13] Khuri, S. (2017). *Lecture Notes on BioInformatics*

[14] Xia, X. (2012). *Position weight matrix, gibbs sampler, and the associated significance tests in motif characterization and prediction. Scientifica, 2012.* Available [Online]: `http://downloads.hindawi.com/journals/scientifica/2012/917540.pdf` Last retrieved: 04/26/2017

[15] Claverie, J. M., & Audic, S. (1996). *The statistical significance of nucleotide position-weight matrix matches. Computer applications in the biosciences: CABIOS, 12(5), 431-439.* Available [Online]: `http://bioinformatics.oxfordjournals.org/content/12/5/431.full.pdf` Last retrieved: 04/26/2017

[16] Pang-Ning, T., Steinbach, M., & Kumar, V. (2006). *Introduction to data mining. In Library of congress (Vol. 74)*

[17] Davis, J., & Goadrich, M. (2006, June). *The relationship between Precision-Recall and ROC curves. In Proceedings of the 23rd international conference on Machine learning (pp. 233-240). ACM.*

Available [Online]: http://machinelearning.wustl.edu/mlpapers/paper_files/icml2006_DavisG06.pdf Last retrieved: 04/26/2017

[18] Boulesteix, A. L., Janitza, S., Kruppa, J., & König, I. R. (2012). *Overview of random forest methodology and practical guidance with emphasis on computational biology and bioinformatics. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 2(6), 493-507.* Available [Online]: https://epub.ub.uni-muenchen.de/13766/1/TR.pdf Last retrieved: 04/26/2017

[19] Liaw, A., & Wiener, M. (2002). *Classification and regression by randomForest. R news, 2(3), 18-22.* Available [Online]: http://ai2-s2-pdfs.s3.amazonaws.com/6e63/3b41d93051375ef9135102d54fa097dc8cf8.pdf Last retrieved: 04/26/2017