FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

FEUP

Application of Machine Learning techniques on the Discovery and annotation of Transposons in genomes

Tiago David Soares da Cruz Loureiro

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Jorge Vieira (PhD)

Supervisor: Nuno Fonseca (PhD)

Supervisor: Rui Camacho (PhD)

July 17, 2012

Application of Machine Learning techniques on the Discovery and annotation of Transposons in genomes

Tiago David Soares da Cruz Loureiro

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

July 17, 2012

Abstract

Transposable elements or transposons are sequences of DNA that move and transpose within a genome. Known as mutation agents, these elements are broadly studied due to their importance in disease research in genome alteration and due to their importance on species evolution.

Several methods were developed to discover and annotate transposable elements. They are usually classified in four main categories: *De novo*; Structure-based; Comparative Genomic; Homology-based.

Although there are different tools, based on these methodologies for detecting transposable elements, there isn't any single tool achieving good results on the detection of all the different types of transposable elements.

The aim of this dissertation is to improve transposon detection by analyzing existing transposon detection tools and evaluate their performance regarding the detection and annotation of transposable elements.

Taking this into account, this dissertation will have three distinct phases. At first there will be generated datasets of curated DNA sequences with transposable elements inserted in known positions. These datasets will be as diversified as possible so they can cover all the different scenarios found in real genomes.

Following, using these datasets, transposon detection tools are evaluated and their accuracy is measured.

Using the results of the transposon detection tools' evaluation, the last step of this dissertation is to use Machine Learning techniques to identify and characterize the context where these tools fail short in transposon detection. After this, the aim is to create a meta-tool using models generated by Machine Learning that combines the best of different transposable elements detection tools in order to improve the overall detection accuracy.

Acknowledgements

It would not have been possible to write this thesis without the help and support of the kind people around me. Above all, I would like to thank my family for their unceasing support and patience. I would also like to acknowledge Rui Camacho, Nuno Fonseca and Jorge Vieira for their support, help and suggestions. Without their precious help and expertise, this work could not be possible.

Tiago Loureiro

"I love fools' experiments.
I am always making them."

Charles Darwin

Contents

List of Figures

List of Tables

LIST OF TABLES

Abbreviations

DNA	Deoxyribo Nucleic Acid
TE	Transposable Element
mtDNA	Mitochondrial DNA
cDNA	DNA copy
BLAST	Basic Local Alignment and Search Tool
Transposase	An enzyme that is responsible for the catalysis of a transposition
MITE	Miniature Inverted Repeat Transposable Element
SINE	Short Interspersed Nuclear Element
LTR	Long Terminal Repeat
TIR	Terminal Inverted Repeat
A	Adenine
G	Guanine
C	Cytosine
T	Thymine
bp	Base Pairs
gag	The genomic region encoding the capsid proteins
pol	The genomic region encoding the viral enzymes protease, reverse transcriptase, and integrase
ENV	The genomic region encoding the viral glycoproteins
FASTA	Text-based format for representing nucleotide sequences, each one represented by a single letter
Indel	A mutation class that includes both insertions and deletions

Chapter 1

Introduction

This Chapter introduces the work of this thesis by presenting its context. It follows by presenting the motivation and the goals of this work. Lastly, it presents the document's structure.

1.1 Context

Transposable elements (TEs) or simply transposons are a large class of repetitive DNA sequences that have the ability to move within a given genome. It is estimated that 40% or more of the human genome is composed of transposon-derived sequences and although many are remnants of active elements, the mobility of transposons has had an important role in the structure and evolution of genes and genomes. Having the ability to replicate, transposons can occupy large fractions of genome sequences.

The identification of TEs has become increasingly necessary for both genome annotation and evolutionary studies. The contribution of TEs to genome structure and evolution as well as their impact on genome sequencing, assembly, annotation and alignment has generated increasing interest in developing new methods for their computational analysis.

Many approaches to detect TEs were developed since transposons were found by Barbara McClintock [McC50]. Based on different properties of this TEs, these approaches look for structural similarities, compare sequences with known transposons or try to identify repetitive patterns.

Given the existence of different transposon detection methodologies, there are several software tools that implement these different approaches. Each tool has its own strengths and weaknesses in the detection of particular TE category. Hence there are specific tools to achieve better detection rates on specific transposon categories. Adding to this, different tools can disagree on the detection of a TE, whether not agreeing on its length or boundaries (beginning position and/or the ending position).

In this perspective, transposon detection tools may have the potential to benefit from comparison between each other and on the integration of multiple tools in a computational dynamic system.

1.2 Motivation and Goals

Transposable elements are very important entities to be studied as they have preponderant roles in genome structure, size, rearrangement and have great contribution to host gene and regulatory evolution. Also they are very useful in plant molecular biology since they are mutation agents and as such they can introduce desirable characteristics where they are inserted.

Due to their importance, various approaches were made in order to identify and annotate transposon elements such as *De novo*, Homology-based, Structured-based and Comparative genomic methods.

De novo methods compare several sub-sequences of a given genome and if they are repeated several times within that genome then they can potencially be TEs. The key step on this approach is to distinguish TEs from all other repeat classes. On Homology based methods sets of known TEs are compared with DNA sequences of a genome to find similarities. Structure based Methods use prior knowledge about the common transposon structural features, such as long terminal repeats to identify potential transposon elements on a given genome. Finally Comparative genomics methods use multiple alignments of closely related genomes to detect large changes between genomes. Although different, all these approaches pursue the detection of TEs and each has its own strengths and weaknesses that will be described in Chapter 3.

Several studies have been published describing and comparing transposon detection tools [BQ07, CD03]. In these studies several transposon detection tools are described and their implementations compared. However the results of transposon detection vary from tool to tool. Transposon detection agreement between different tools is one of the main problems since tools can identify or not a given transposon and even if identified they can disagree on its length and start and end positions within the genome.

On another perspective, and according to [BQ07] and [CD03], integration of multiple approaches will further advance the computational analysis of this dynamic component. This means that integrating the best of each relevant tool can improve the overall detection accuracy and provide researchers better result in transposon detection.

The aim of this dissertation is firstly to assess the performance of existing TE detection tools. Secondly, using the gathered data, increase the performance of the analyzed TE tools by combining their predictions using machine learning models. This approach aims to take advantage of the strengths of each one of the analyzed tools on different TE discovery contexts.

With this in mind, the main idea is to use datasets of DNA sequences that have transposon sequences present in *a priori* positions. Using these datasets, several discovery methodologies are tested and evaluated accordingly to the expected results of the datasets. Using the results of the previous step as a data source, machine learning techniques will be used in order to combine different transposon detection methodologies and create a model to increase the accuracy of transposon element detection and annotation.

1.3 Document Structure

The next Chapter provides a background of transposon related biological information and gives an insight on the Machine Learning subject.

In chapter 3, state of the art on transposon is presented and some discovery approaches will be described and some considerations will be made about the strength of them in finding particular types of transposons elements. Transposon detection tools will also be analyzed later in this chapter.

Chapter 4 summarizes the technologies that will be used on this project and describes the experimental setup, validation and evaluation of the results of this work.

Chapter 5 describes the machine learning process over the gathered data and analyzes the results obtained.

Chapter 6 concludes this report and presents future possibilities for improvement.

Chapter 2

Background

In this chapter several concepts related to transposable elements (TEs) and genomes are introduced. The concept of TEs is presented and their importance in the genome is emphasized. The main kinds of TEs are presented and characterized.

Finally, at the end of the chapter, we draw some conclusions regarding the transposable elements and their discovery methodologies.

2.1 Transposable Elements

Initially discovered during the middle part of the twentieth century by Barbara McClintock, transposable elements (TEs), "jumping genes" or simply transposons are DNA sequences that move from one genome location to another. According to the The American Heritage Science Dictionary [Ame05] the definition of transposon is the following:

> 'A segment of DNA that is capable of independently replicating itself and inserting the copy into a new position within the same or another chromosome or plasmid.'

McClintock's work [McC50] was revolutionary in that it suggested that an organism's genome is not a stationary entity, but rather it is subject to alterations and rearrangements. Due to this major contribution to Science she was awarded the Nobel Prize in 1983.

2.1.1 DNA sequences

Firstly published in 1953 by Watson and Crick [WC53], DNA stands for deoxyribonucleic acid. It is the genetic material which makes up all living cells and many virus. It consists of two long strands of nucleotides linked together in a helicloidal structure. In eukaryotic cells, the DNA is contained in the nucleus, where it is called nuclear DNA, and in mitochondria where it is called mitochondrial DNA or mtDNA.

DNA has asymmetric ends called the 5' (five prime) and 3' (three prime) ends, with the 5' end having a terminal phosphate group and the 3' end a terminal hydroxyl group.

The information in DNA is stored as a code made up of four chemical bases: Adenine (A); Guanine (G); Cytosine (C); Thymine (T). The order, or sequence, of these bases determines the information available for building and maintaining an organism. DNA bases pair up (adenine with thymine and cytosine with guanine) to form units called base pairs (bp), as shown in Figure 2.1. Each base has a five-carbon sugar, deoxyribose, and one phosphate group attached. This group forms a nucleotide which is present in the form of a spiral called double helix [Ref12].

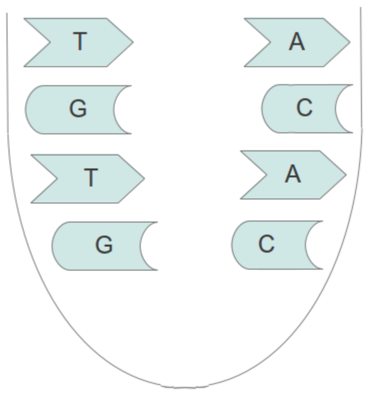

Figure 2.1: DNA structure

The DNA segments carrying genetic information are called genes.

2.1.2 Gene

Genes are genomic sequences that represent the basic physical and functional unit of heredity. Genes, which are made up of DNA, act as instructions to make molecules called proteins.

Although the classic view of genes defines them as compact, information-laden gems hidden among billions of bases of junk DNA, they are neither compact nor uniquely important. According to [Pen07] and [GP08] genes can be sprawling, with far-flung protein-coding and regulatory regions that overlap with other genes.

2.1.3 Mutations

Mutations are sudden and spontaneous changes in genome sequences caused by radiation, virus, transposons and mutagenic chemicals, as well as errors that occur during meiosis or DNA replication. TEs, as integral elements of genomes are possible targets of these mutations.

Mutations in small scale can be classified as:

- Point mutations: exchange a single nucleotide for another, these changes are classified as transitions or transversions. Most common is the transition that changes an A for a G or a G for an A or a T for a C or a C for a T. A transition can be caused by nitrous acid, base mis-pairing, or mutagenic base analogs. Less common is a transversion, which exchanges a C/T for A/G. A point mutation can be reversed by another point mutation, in which the nucleotide is changed back to its original state or by second-site reversion (a complementary mutation elsewhere that results in regained gene functionality). Point mutations that occur within the protein coding region of a gene may be classified into three kinds, depending upon what it codes for:

  - Silent mutations: which code for the same amino acid.
  - Missense mutations: which code for a different amino acid.
  - Nonsense mutations: which code for a stop and can truncate the protein.

- Indel mutations [KR04]:

  Insertions are additions of one or more extra nucleotides into a DNA sequence. It may be caused by TEs or by errors during replication of transposons. The consequences of an insertion in a coding region of a gene may be altering the splicing of the mRNA (a modification of an RNA after transcription, in which introns are removed and exons are joined) or causing a shift in the reading frame. These insertions can be reverted by excision of the TEs element.

  Deletions are remotions of one or more nucleotides from a DNA sequence and, as a consequence, they can alter the reading frame of the gene in which they happen.

2.1.4 Importance of Transposable Elements

The fact that more than 40% of human genome is made up of TEs raises an important question concerning their importance in the genome [nat01].

TEs can be the source of genome construction and at the same time of their destruction [BHD08]. TEs can damage the genome of their host cell in different ways, namely by:

- Insertions: TEs can insert themselves into functional genes and disable them. This process can cause numerous diseases depending on the TE. Among these diseases are hemophilia A and B, colon Cancer or even Cystic fibrosis. An example can be seen in Figure 2.2.

- Deletions: if a transposon leaves a gene, the resulting gap may not be repaired correctly.

- TE multiplication: multiple copies of the same sequence can hinder precise chromosomal pairing during mitosis and meiosis, resulting in unequal crossovers, one of the main reasons for chromosome duplication.

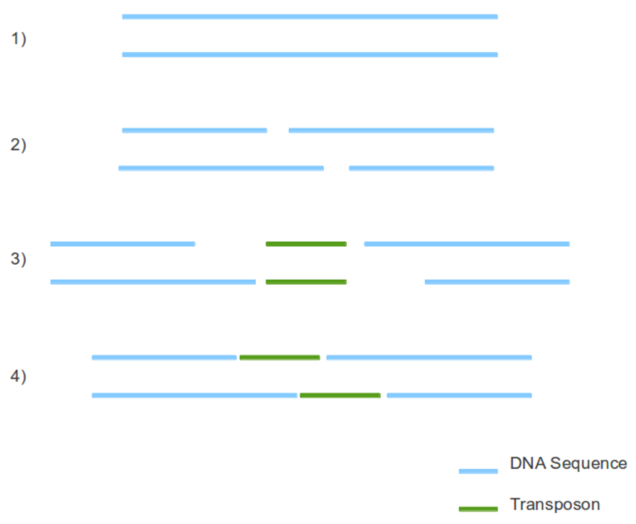

Figure 2.2: Example of an insertion of a TE in a DNA segment in four steps

On the other hand, insertion of transposons is accompanied by the duplication of a short flanking sequence of a few base pairs. Transposons excise imprecisely, generally leaving part of the duplication at the former insertion site and the consequences of the insertion and excision depend on the location within the coding sequence. This commonly results in either an altered gene product or a frame-shift mutation. Ultimately there can be generated an exon shuffling. Exon (coding section of an RNA transcript, or the DNA encoding it, which are translated into a protein) shuffling results in the juxtaposition of two previously unrelated exons, usually by transposition, thereby potentially creating novel gene products [MDK99].Transposons also can promote illegitimate recombination of large DNA segments movement of large segments of DNA either by transposition or by illegitimate recombination [FRI86].

TEs are not all active. In fact, most TE are silent and do not actively move around the genome in which they are inserted. Some silented TEs are inactive because they have mutations that affect their ability to move from one chromosomal location to another while others are perfectly intact and capable of moving but are kept inactive by defense mechanisms such as DNA methylation, chromatin remodeling, and miRNAs [Pra08].

Transposons have, therefore, the ability to increase genetic diversity. Adding to this the fact that most TE are inhibited by the genome, results in a balance that makes TEs an important part of evolution and gene regulation in all organisms that carry these sequences.

2.1.5 Transposable Elements Classification

In the past, TE classification was based on a wide variety of factors. Although useful, this approach was limited due to the fact that many of the newly identified transposons did not contain the structural elements signature that are found in the earlier classes of transposon. In an era of large-scale genome sequencing, in which new elements are being described from diverse organisms at an unprecedented rate, a better way of categorizing TEs is by how they transpose [CD03].

In an attempt to introduce a universal classification scheme, Wicker *et al* [WSHV+07] defined a classification scheme based on the transposition process [KJ08] and implemented in Repbase [Rep], which is a large database of eukaryotic repetitive elements and TEs.

Using this criteria, transposons are first assigned in one of two classes. If their mechanism of transposition is of "copy and paste" they are classified as Retrotransposons. If the mechanism of transposition is of "cut and paste" they are classified as DNA transposons. The way they move within a genome is dictated by the corresponding transposase proteins which can be: DDE-transposases, rolling-circle (RC) or Y2-transposases, tyrosine (Y)-transposases, serine (S)-transposases and a combination of reverse transcriptase and endonuclease (RT/En). Transposases are DNA-binding enzymes that catalyze “cut and paste” or “copy and paste” reactions to promote the movement of DNA sequences. Depending on the transposase used in the transposition process, TEs belong to one of five major classes: long terminal repeat (LTR) retrotransposons, non-LTR retrotransposons, cut-and-paste DNA transposons, rolling-circle DNA transposons (Helitrons) and self-synthesizing DNA transposons (Polintons). Each of these classes of TEs are composed by a small number of superfamilies or clades and each superfamily is composed by many families of TE [KJ08].

2.1.5.1 Retrotransposons

Retrotransposons, Class I transposons or simply "copy and paste" generate a copy of their DNA (cDNA) by reverse transcription of their RNA genome. The cDNA is then inserted into the genome in a new position. Reverse transcription is catalyzed by a reverse transcriptase, which is often coded by the TE itself. These elements can be divided in three classes according to the enzymatic activities they encode in addition to reverse transcriptase [CCGL02].

Long Terminal Repeats (LTR) and retrovirus have adopted DDE-transposases to integrate the cDNA. The transcription process described in Figure 2.3.

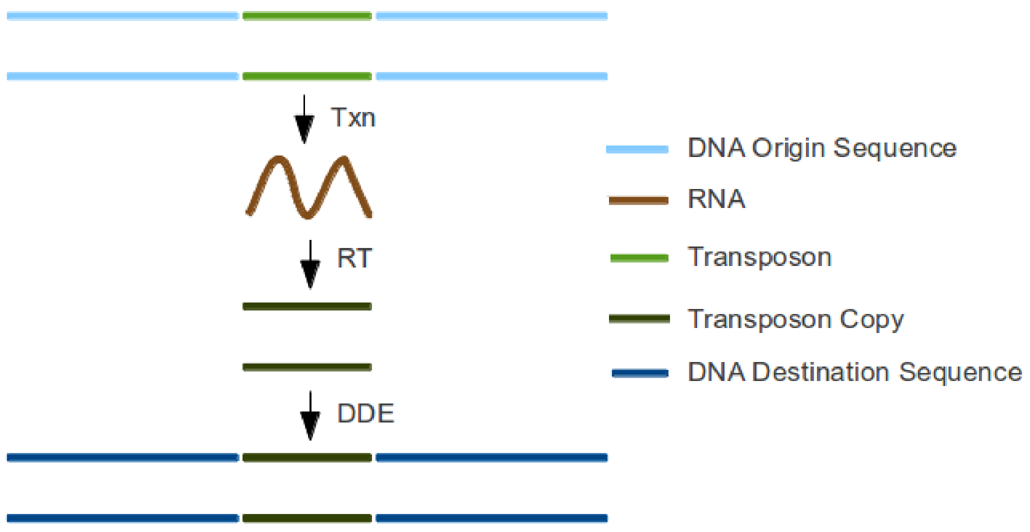

Figure 2.3: Retrotransposons copy-out themselves by reverse-transcribing a full-length copy of their RNA, generated by transcription (Txn). They make a cDNA from their RNA and integrate this into a target using a DDE-transposase.

The Long Terminal Repeats are two normally homologous non-coding DNA sequences that flank the internal region of certain mobile genetic elements and that usually begin and end in 5'-TG CA-3' inverted repeats [PBV02]. The internal region contains the characteristics *gag* and *pol* (and *env* in the case of retroviruses) ORFs, which encode for the different protein products required for the replication cycle and transposition of LTR retrotransposons. LTR retrotransposons can be either autonomous or non-autonomous depending on their need of other mobile-related protein for the transposition process.

Autonomous LTR retrotransposons can be classified into four major groups or families based on sequence similarity: Ty1/Copia, the Ty3/Gypsy, the Bel/Pao, and the Retroviridae families.

1. Ty1/Copia elements are abundant in all species. The Ty1/Copia internal region encodes for the typical gag and pol polyproteins but differ from other LTR retrotransposons in the ORF position of integrase (INT), which maps after to protease (PR).

2. Ty3/Gypsy LTR retroelements also spread in animal, plant and fungal organisms.

3. Bel/Pao LTR retroelements have been only described in metazoan genomes. This family includes both LTR retrotransposons and retroviruses, as it is now known that some Bel/-Pao species encode for putative env-like genes. These LTR retroelements show the same genomic ORF organization than Ty3/Gypsy family and are taxonomically known as semotiviruses.

4. Retroviridae constitute a family of retroviruses which specifically inhabit or circulate in the genomes of vertebrates. The *gag-pol-env* genome ORF organization of the Retroviridae is similar to that of Ty3/Gypsy and Bel/Pao simple retroviruses. In most cases the multiple distinct Retroviridae species are complex retroviruses that incorporate in their genomes some additional accessory genes which are necessary for the replication cycle and transmission from a host into another.

Non-autonomous LTR retrotransposons show LTRs but usually lack of coding capacity and have been only described in plants. There are two main classes of non-autonomous LTR retrotransposons: the LArge Retrotransposons Derivatives (LARDs) and the Terminal-repeat Retrotransposons In Miniature (TRIMs) [KVP+04].

LARDs range from 5.5 kb to 8.5 kb and have large-conserved-no coding internal domains of 3.5 kb flanked by long LTRs (4.5 kb). According to Kalendar *et al.* [KVP+04] LARDs evolve from Ty3/Gypsy LTR retroelements as they apparently use in the replication and integration process distinct protein products encoded by their putative Ty3/Gypsy LTR retroelements.

TRIMs elements are less than 540 bp in size. They show a small non-coding central domain of 100–300 bp which contains PBS and PPT motifs and that is flanked by small LTRs of 100–250 bp in size. These elements are involved actively in the restructuring of plant genomes [WLBK01].

The DIRS1 class of retrotransposons uses Y-transposases and does not have LTRs. The insertion process is described in Figure 2.4.

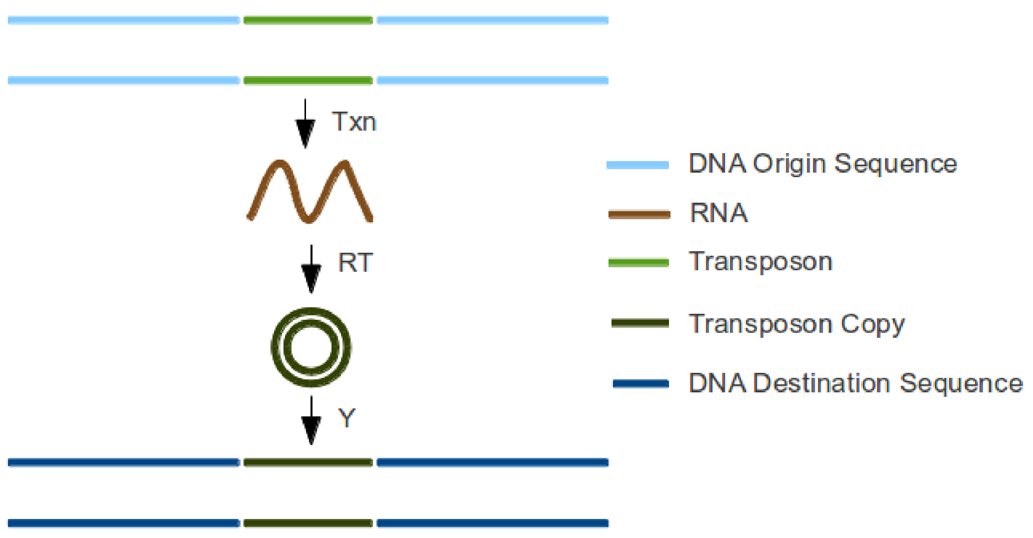

Figure 2.4: Y-retrotransposons copy-out themselves by reverse-transcribing a full-length copy of their RNA, generated by transcription (Txn). They generate a circular cDNA intermediate by reverse transcription and a Y-transposase integrates the element into the target.

The typical structure of DIRS elements contains inverted terminal repeats (ITRs), internal ORFs and an internal complementary region (ICR) derived from the duplication of flanking ITR sequences.

The third class of is non-LTR-retrotransposons, also called TP-retrotransposons that use a combination of reverse transcriptase and endonuclease (RT/En) to transpose. These transposons lack terminal repeats but often have A-rich sequence at their 3' end. The process of their transposition is described in Figure 2.5.

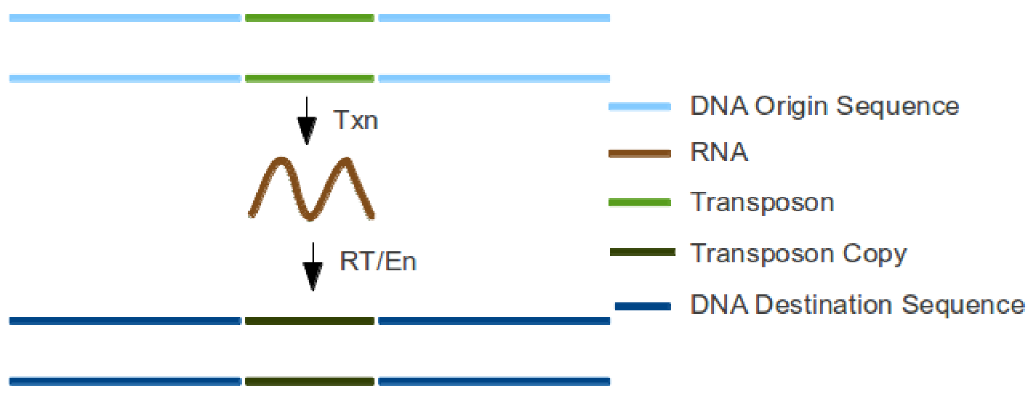

Figure 2.5: Non-LTR-retrotransposons use reverse transcriptase to copy their RNA directly into a target that has been cut by a transposon-encoded nuclease.

Autonomous non-LTR retroelements have been grouped in two major classes:

- R2 elements: these elements encode for a single ORF with a central RT domain and an endonuclease conserved domain.
- Long INterspersed repetitive Elements (LINEs) : these are a family of 6-8 kb long elements that encode reverse transcriptase and lack LTRs. They are transcribed to an RNA using an RNA polymerase II promoter that resides inside the LINE [Ala02].

Non autonomous non-LTR retroelements are DNA sequences of 80 to 630 bp known as Short INterspersed repetitive Elements (SINEs). They don't encode a functional reverse transcriptase protein that are transcribed by RNA polymerase III [Ala02]. For this reason it has been proposed that SINEs use the enzymatic machinery of LINEs for their retrotransposition [KBWRE09].

2.1.5.2 DNA transposons

DNA transposons, Class II transposons or simple "cut and paste" do not involve an RNA intermediate in the transposition process. The transposase makes a staggered cut at the target site producing sticky ends, cuts out the DNA transposon and ligates it into the target site. A DNA polymerase fills in the resulting gaps from the sticky ends and DNA ligase closes the sugar-phosphate backbone. This results in target site duplication and the insertion sites of DNA transposons may be identified by short direct repeats followed by inverted repeats [CD03].

Due to the nature of the transposition process, these transposons have unique characteristics:

1. Create target site duplications upon insertion;
2. Contain an ORF containing the catalyst domain for transposase;
3. Have Terminal Inverted Repeats (TIRs) in the extremities.

TIRs sequences contain DNA-binding sites that are recognized by the Transposase to form the synaptic protein-DNA complex and thus facilitating the cleavage at both strands and the subsequent transposon release.

DNA transposons can be classified as either "autonomous" or "non-autonomous". The autonomous ones have an intact gene that encodes an active transposase enzyme; the TE does not need another source of transposase for its transposition. In contrast, non-autonomous elements encode defective polypeptides and accordingly require transposase from another source.

Miniature Inverted-repeat Transposable Elements (MITEs) are non-autonomous DNA transposons that are relatively small and share the TIR sequence motifs with other DNA transposons. They are abundant in the non-coding regions of the genes of many plant and animal species [HW10].

DNA transposons may be duplicated if transposition takes place during S phase of the cell cycle when the donor site has already been replicated, but the "target" site has not.

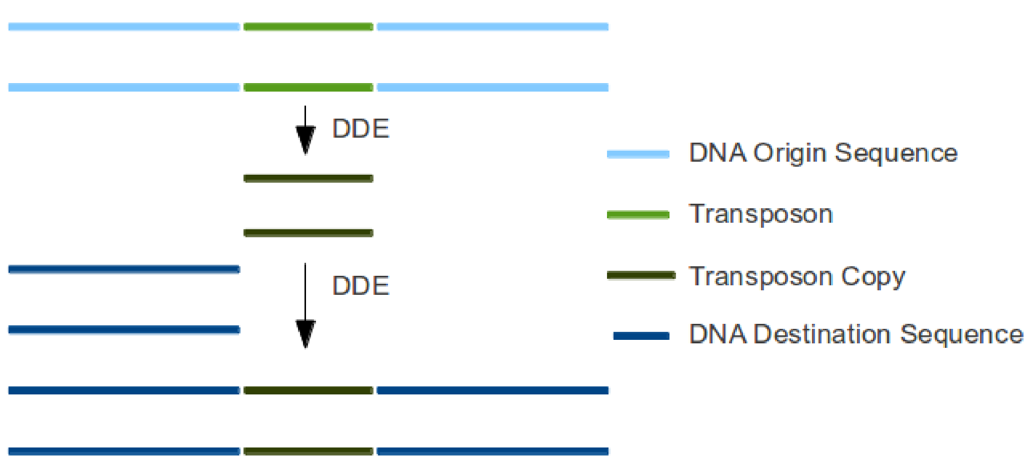

Figure 2.6: In general, DNA transposons excise from the flanking DNA to generate an excised linear transposon, which is the substrate for integration into a destination sequence.

2.2 Machine Learning

We now describe some basic concepts in Machine Learning and its applicability in Data Mining problems. The algorithms more relevant for this dissertation are characterized and described.

2.2.1 Introduction

Solving complex computing problems does require intelligence. The learning process is crucial in building intelligence as it is a source of knowledge and it is in this context that the Machine learning concept becomes relevant.According to the Tom M. Mitchell [Mit97] definition, a Machine Learning program can be described as follows:

> 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.

Machine learning is a research area within artificial intelligence that develops algorithms that give the capability of acquiring and integrating new knowledge automatically.

In the acquiring process, it is expected that it is used a set of examples (training data) from which it can extract regularities.

2.2.2 Datasets as Machine Learning inputs

A Machine Learning method takes as an input a training dataset in which it will work on. In this dataset it is needed to clarify four different concepts [WF05]:

- Instance: a single example in a dataset. For e.g., a row in table 2.1.

- Attribute: an aspect of an instance, also called feature. For example Outlook, Temperature, Humidity in table 2.1. Attributes can take categorical or numeric values.
- Value: category that an attribute can take. For instance Sunny, Rainy for the attribute Outlook in table 2.1.
- Concept: the aim, the label to be learned. In table 2.1, the answer to Play Sport (Yes/No).

Outlook	Temperature	Humidity	Play Sport?
Sunny	Warm	Low	Yes
Sunny	Cold	Medium	No
Rainy	Cold	Low	No
Sunny	Hot	Medium	Yes
Rainy	Hot	High	Yes
Rainy	Cold	High	No

Table 2.1: Sample table

2.2.3 Algorithm Types

According to [RN03], the type of feedback available for learning is usually the most important factor in determining the nature of the learning problem that the agent faces. The field of machine learning usually distinguishes three cases: supervised, unsupervised, and reinforcement learning.

- Supervised learning: is a learning process from which machine learning can infer a function from supervised (labeled) training dataset. Each example in this training dataset is formed by an input object with one or more attribute values and a desired output value. These examples are analyzed and it is produced an inferred function, which is called a classifier or a regression function depending on the output being discrete or continuous. The result function should predict the correct output given any valid input and is evaluated regarding it's accuracy of predicting correct outputs.
- Unsupervised learning: is a learning process which tries to learn relations between data components or learning patterns with no specific output values (no labeled data). Since the examples given to the learner are unlabeled, there is no error or reward signal to evaluate a potential solution and therefore this learning process cannot learn what to do due to the fact that there is no information on what is or isn't a desirable output.
- Reinforcement learning: is a learning process which is based on learning from reinforcement. It is based on the notion of maximizing the decisions value by some notion of cumulative reward. Reinforcement learning typically includes the subproblem of learning how the environment works.

2.2.4 Machine Learning Techniques

Following several popular approaches of Machine Learning are presented based on different types of machine learning types described in section 2.2.3.

2.2.4.1 Decision Tree learning

Decision tree learning is a method commonly used in data mining, statistics and machine learning for approximating discrete-valued target functions in which the learned function is represented by a decision tree. The goal of this learning method is to create a model that predicts the value of a target variable based on several input variables. Decision trees can be on of two main types:

- Classification trees: when the predicted outcome is the class to which the data belongs.
- Regression trees: when the predicted outcome can be considered a real number.

Decision trees classify instances by sorting them through a tree. Each node in the tree specifies a test of some attribute of the instance and each branch derived from that node corresponds to one of the possible values for that attribute. Starting in the root node of the tree, the classification is achieved when it reaches a leaf with no more branches [Mit97].

Figure 2.7 illustrates a typical decision tree that classifies a weather condition suitable or not for playing a sport.

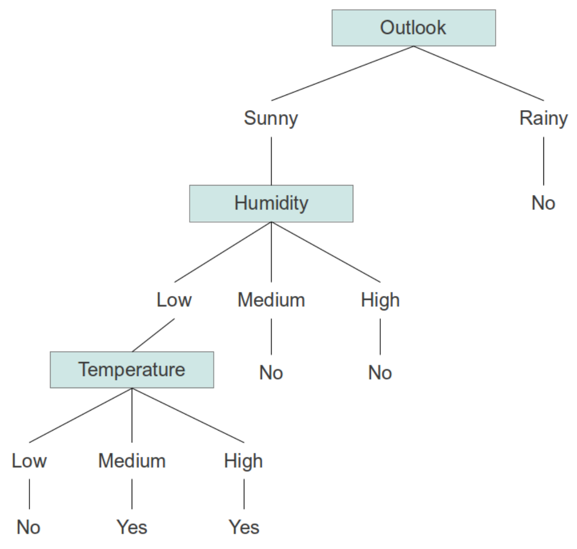

Figure 2.7: Example of a decision tree for playing a sport

To build a decision trees there are several appropriate algorithms. Among them, the most recognized are:

- ID3 Algorithm: Iterative Dichotomiser 3 (ID3) is an algorithm used to generate a decision tree created by Quinlan, J. R. [Qui86]. This algorithm constructs the decision tree from top to bottom, always using the best attribute to be tested on each node in the tree. In fact, the central choice in ID3 algorithm is selecting the most useful attribute for classifying examples. Once chosen the attribute for a node, the descendant nodes are created for each possible values of that attribute and the training examples are sorted to the appropriate descendant node.

Smaller decision trees should be preferred over larger ones as they generalize better. This algorithm usually produces small trees, but it does not always produce the smallest possible tree. The optimization step makes use of information entropy as it it show in 2.1.

$$Entropy(S) = -\sum_{j=1}^{n} fs(j)\log_2 fs(j) \tag{2.1}$$

where:

- Entropy(S) is the information entropy of the set S;
- n is the number of different values of the attribute in S;
- fs(j) is the frequency of the value j in S.

The entropy is used to find which node is to split next in the algorithm: if the value is 0, it is perfectly classified and the bigger this value is the higher the potential to improve the classification in that node.

Using the entropy as a measure of the impurity in a collection of training examples, it is possible to use a measure of the effectiveness of an attribute in classifying a training data: the information gain which is defined in 2.2.

$$Gain(S,A) = Entropy(S) - \sum_{v=Values(A)} \frac{|Sv|}{|S|} Entropy(Sv) \tag{2.2}$$

where:

- Entropy(S) is the information entropy of the set S;
- Values(A) is the set of all possible values for attribute A;
- Sv is the subset of S for which attribute A has value v.

Gain quantifies the entropy improvement and helps ID3 algorithm achieve better results.

- C4.5 Algorithm: C4.5 is an extension of Quinlan's earlier ID3 algorithm [Qui93]. This algorithm builds decision trees from a set of training data in the same way as ID3, using the concept of information entropy. At each node of the tree, C4.5 chooses one attribute of the data that most effectively splits its set of samples into subsets enriched in one class or the other.

> The algorithm works as follows. It first checks if all the samples in the list belong to the same class. When this happens, it creates a leaf node for the decision tree saying to choose that class. If none of the features provide any information gain it creates a decision node higher up the tree using the expected value of the class. If it encounters an instance of previously-unseen class it creates a decision node higher up the tree using the expected value. Having these rules as a base, it calculates the normalized information gain for each attribute from splitting on a given point. The highest normalized information gain attribute is used to create a node and split. It them processes the sublists obtained by splitting on that node and adds them as children.
>
> C4.5 brings many improvements over the ID3. Some of them are the ability to handle both continuous and discrete attributes, handling training data with missing attribute values which are not used in gain and entropy calculations, handling attributes with differing costs and it prunes the tree after creation in an attempt to remove branches that do not help by replacing them with leaf nodes.

Decision tree learning can handle numeric and categorical data and create easy to understand and interpret models. On the other hand, decisions tree can fail to generalize data and may require pruning.

2.2.4.2 Association Rule learning

Association rule learning is a popular method for discovering interesting relations, frequent patterns, associations, correlations, or causal structures among sets of items in large databases.

According to [BJ93] the problem of association rule mining is defined as the follow:

Let I be a set of n binary attributes called items and D be a set of n transactions called the database. Each transaction in D has a unique transaction ID and contains a subset of the items in I. A rule is defined as an implication of the form X $\rightarrow$ Y where X and Y belong to I. X is called antecedent and Y is called the consequent of the rule.

To select interesting rules from the set of all possible rules, various measures of significance and interest can be used to evaluate their pertinence and the their real value. The most used are:

- Support: denotes the frequency of the rule within transactions. A high value means that the rule involves a great part of database.

$$Support(A \rightarrow B) = p(A \cup B) \tag{2.3}$$

- Confidence: denotes the percentage of transactions containing A which also contain B. It is an estimation of conditioned probability.

$$Confidence(A \rightarrow B) = p(B|A) \tag{2.4}$$

- Lift: denotes the ratio of the observed support to that expected if A and B were independent.

$$Lift(A \rightarrow B) = \frac{p(B|A)}{p(B)} \tag{2.5}$$

2.2.4.3 Artificial neural networks

Artificial neural networks are inspired by networks of biological neurons. They contain layers of simple computing nodes that operate as nonlinear summing devices. These nodes are richly interconnected by weighted connection lines, and the weights are adjusted when data are presented to the network during the training process. Neural networks can perform tasks such as predicting an output value, classifying an object, approximating a function, recognizing a pattern or completing a known pattern [Jud01].

Neural networks are based on the concept of neurons (simple units, or nodes). Each neuron takes a number of real-valued inputs and produces a single real-valued output. There are different weights that are assigned to each input link of the neuron and the input link values are multiplied by them and then summed. After summing all the input combinations there is used an activation function that receives this input and returns the output of the neuron to the given input. An example of a neuron can be seen in figure 2.8.

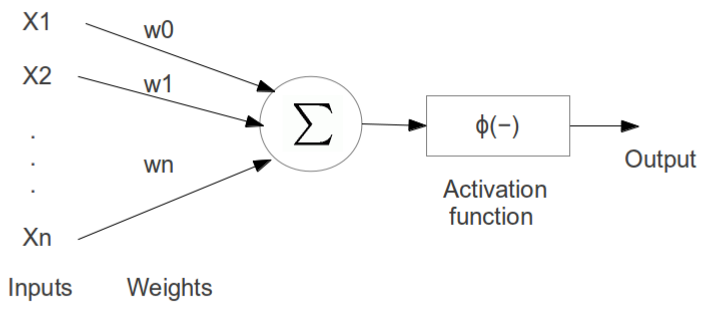

Figure 2.8: Neuron example.

Neural networks can have different architectures, mostly they are divided in two categories: single-layer feed-forward and multi-layer feed-forward. In the first category, the input layer of nodes is directly connected to the output layer of neurons while on the second there is one or more hidden layers of nodes between the input and output nodes.

To train a neural network initially it is chosen small random weights and a small learning rate. Next each member of the training set is applied to the neural network using a training rule to adjust the weights. For each neuron is computed the network input, the output value of the activation function and it is computed the error and weights are updated. The training rule depends on the neural network architecture.

The main advantages of using neural networks are their ability to be used as an arbitrary function approximation mechanism that is continuously learning from observed data. They have generally an high prediction accuracy and they work well even if the training dataset contains errors or noisy data.

Tunning the learning algorithms to achieve better results can take significant amount of experimentation and if the model chosen is overly complex, neural networks tend to have problems in learning.

2.2.4.4 Inductive logic programming

Inductive logic programming (ILP) is an approach to rule learning using logic programming as a uniform representation for examples, background knowledge, and hypotheses. Based on known background knowledge and a set of examples represented as a logical database of facts, an ILP system will find hypotheses of a certain format that can predict the class labels of target tuples. Although many ILP approaches achieve good classification accuracy, most are not highly scalable due to the computational expense of repeated joins [MdR94].

2.2.4.5 Support vector machines

Support vector machines are a set of related supervised learning methods used for classification and regression. They use a nonlinear mapping to transform the original training data into a higher dimension. Within this new dimension, it searches for the linear optimal separating hyperplane, which is a boundary that separates the tuples of one class from another. With an appropriate nonlinear mapping to a sufficiently high dimension, data from two classes can always be separated by a hyperplane. The SVM finds this hyperplane using support vectors and margins.

2.2.4.6 Bayesian networks

A Bayesian network is a probabilistic graphical model that represents a set of random variables and their conditional dependencies via a directed acyclic graph. The nodes of the graph represent random variables which may be observable quantities, latent variables, unknown parameters or hypotheses and edges represent conditional dependencies. Each node is associated with a probability function that takes as input a particular set of values for the node's parent variables and gives the probability of the variable represented by the node [WF05].

Building a learning algorithm for Bayesian networks requires the definition of two main components: an evaluation function to evaluate the network on the data and a method for searching through the space of all possible Bayesian networks. To evaluate the quality of a network, two popular measures are commonly used:

- Akaike Information Criterion score

$$AICscore = -LL + K \tag{2.6}$$

- Minimum Description Length score

$$MDLscore = -LL + \frac{K}{2}\log N \tag{2.7}$$

where LL is the log-likelihood, which is the sum of the logarithms of the probabilities of the network accords to each instance, K the number of parameters and N the number of instances in data.

To the search in the space of possible Bayesian networks, algorithms such as Greedy hill-climbing, Simulated annealing or Genetic algorithms can perform this task.

There are various algorithms capable of learning Bayesian networks. Among them, the most recognized are K2 and TAN.

K2 [CD92] starts with a given ordering of the nodes and then it processes each node in turn. It greedily considers adding edges from previously processed nodes to the current one, adding in each step the edge that maximizes the score given by the evaluation function. When there is no improvement, the algorithm turns to the next node.

TAN (Tree augmented Naive Bayes) [FGG97] uses the Naive Bayes classifier and adds edges to it. The class attribute is the single parent of each node of a network and TAN considers adding a second parent to each node. If the class node and all its edges are excluded from consideration and if there is exactly one node to which a second parent is not added, the resulting classifier has a tree structure rooted at the parentless node.

2.2.5 Evaluating Learning Performance

One subject of great importance in the Machine Learning process is the evaluation of the performance. Evaluation is the key to determine which is the best solution to a particular problem.

There are several measures to evaluate the fitness of models, namely:

- Accuracy: the proportion of correct predictions, both true positives and true negatives in the set of examples considered as defined in 2.8.

$$accuracy = \frac{truepositives + truenegatives}{Totalnumberofexamples} \tag{2.8}$$

- Precision: the proportion of outcomes predicted correctly in all the outcomes predicted as defined in 2.9.

$$precision = \frac{CorrectclassifiedexamplesasX}{TotalexamplesclassifiedasX} \tag{2.9}$$

- Recall: the proportion of outcomes that are correct and were, in fact, retrieved as defined in 2.10.

$$recall = \frac{CorrectclassifiedexamplesasX}{Totalexamplesclassified} \tag{2.10}$$

- F-measure: a weighted harmonic mean of precision and recall as defined in 2.11.

$$f - measure = \frac{2 * precision * recall}{precision + recall} \tag{2.11}$$

While high recall means that an algorithm returned most of the relevant results, high precision means that an algorithm returned more relevant results than irrelevant.

It is often need to compare two different learning methods on the same problem to see which is the better one to use. The model should be able to generalize, deal with new data and do not overfit.

In order to evaluate this, an approach partitions the data into a training and test sets. The model is trained using one partition (training set). As the model is created, it should be tested with the training and the test sets and evaluation metrics referenced in **??** should be calculated. The results of the evaluation using the training set and the test set are then compared to measure the model's ability to generalize.

Since this process requires a training and a test sets, one possible approach is using Cross-validation. Cross-validation is an approach to predict the fit of a model to a hypothetical validation set when an explicit validation set is not available. It optimizes the model parameters to make it fit the training data as well as possible so the model can fit well in the test data, avoiding the overfitting problem. Overfitting is when a model works best for the training data than for the test data and therefore the result model is not able to generalize well. This problem often happens when the size of the training set is small or when there are many parameters in the model.

The first step in the cross-validation involves partitioning the input data in two subsets: the training and the test sets. In k-fold cross validation, this process is then repeated k times (the folds). The input dataset is randomly partitioned into k subsets and of the k subsets, one is used as the validation data for testing the model, and the remaining k-1 are used as training data, with each of the k subsamples being used only once as the validation data. The k results from the folds are then compared. The most common fold number is 10 [WF05].

2.3 Conclusions

This Chapter presented some background information regarding the TEs, mainly support concepts for the aim of this project. It followed by giving a general overview of what are Machine Learning techniques and their importance in the scope of this work.

The next Chapter, chapter 3, uses the definitions and the background introduced in these sections to introduce transposon detection methodologies and tools.

Chapter 3

State of the Art

This chapter describes several studies and approaches concerning the discovery and annotation of TEs in genome sequences. It reviews the principles of transposon discovery and the different methods available. The technologies and tools, which will be used further in this work, are also described and their choice justified.

Section 3.2 describes the main TEs detection methodologies. How they discover and what do they need to do the discovery. Section 3.2 describes transposon detection tools.

3.1 Transposon Detection Methodologies

A recent work of Saha et al. [SBMP08] classifies the TEs detection methodologies based upon the algorithms they use to identify TEs, in three categories:

- Library-based Techniques: identify known repetitive sequences by comparing input datasets against a set of known TEs;
- Signature-based Techniques: identify TEs based on amino acids sequence and spacial arrangements based on *a priori* knowledge of particular transposon types;
- Ab initio Repeat Techniques: identify repetitive elements without using reference sequences or known transposons in the process. It is a two step process where first the algorithms identify repetitive sequences and then they identify the boundaries of the repeats and extraction of the consensus sequence for each family;

An alternative classification is proposed by Bergman and Quesneville [BQ07]. They classify the methods in four categories:

- *De novo*: this approach looks for similar sequences found at multiple positions within a sequence;
- Homology-based: uses known TEs as a mean to discover TEs in genome sequences.

- Structure-based: this approach finds TEs using knowledge from their structure;
- Comparative Genomics: compares genomes to find insertion regions which can be TEs or caused by TEs;

In this work we will follow this last classification scheme.
In order for a tool to be suitable for our project, it had to be:

1. An open source tool and publicly available;
2. Runnable from command-line;
3. Supported by scientific studies.

We have considered tools that meet all of the above conditions. Table 3.1 shows a set of different available TE detection tools.

Name	**Type**	**Operating System**	**URL**	**Citations**
BLAT[Ken02]	Homology - based	Unix and online	http://www.girinst.org/censor/download.php	1949
CENSOR[JKDP96]	Homology - based	Unix and online	http://users.soe.ucsc.edu/~kent/src/	228
LTR_Finder[XW07]	Structure - based	Unix	http://tlife.fudan.edu.cn/ltr_finder/	62
MGEScan[RT09]	*De novo*	Unix	http://darwin.informatics.indiana.edu/cgi-bin/evolution/nonltr/nonltr.pl	4
MITE-Hunter[HW10]	Structure-based	Unix	http://target.iplantcollaborative.org/mite_hunter.html	6
PILER[EM05]	*De novo*	Unix	http://www.drive5.com/piler/	116
RECON[BE02]	*De novo*	Unix	http://selab.janelia.org/recon.html	110
REPEATFinder[VHS01]	*De novo*	Unix	http://cbcb.umd.edu/software/RepeatFinder/	5
RepeatMasker[Smi]	Homology - based	Unix and online	http://www.repeatmasker.org/	6
TEseeker[KUC+11]	Homology - based	Unix	http://repository.library.nd.edu/view/16/index.html	0
VariationHunter[HHD+10]	Structure - based	Unix	http://compbio.cs.sfu.ca/strvar.html	25

Table 3.1: Transposable Elements Detection Tools

Each method is now described in detail.

3.1.1 *De novo*

De novo methods are based on the assumption that similar sequences found at multiple positions within a genome have a strong possibility of being TEs. As such, the main challenges for *de novo* methods are to distinguish TEs from all other types of repeated elements. This intrinsic repetition of transposons in genome sequences makes these methods very effective in finding TEs with high prevalence and not very effective in identifying degraded TEs or transposons with low copy number inside a genome (one or two copies) [KUC+11].

Initially these methods start by using computational strategies such as suffix trees or pairwise similarity to detect repeated regions. With these strategies they aim at detecting pairs of similar sequences at different locations in a given genome. Typically *de novo* repeat discovery methods use assembled sequence data, and therefore are critically dependent on both sequencing and assembly strategies.

The second step in *de novo* approach is to cluster pairs of aligned DNA segments into repeat families and filter out the ones that are not TEs. To filter out the false-positives clusters which correspondes to repetitive elements that are not transposons, there are two different approaches. The first approach considers the multiple alignments of all repeat copies of a cluster. Multiple alignments are split into 'sub-clusters' when several sequences end at the same, or similar, position in the alignment, as expected for interspersed repeats that arise by the process of transposition. Sequences are split according to these boundaries and the underlying alignment pairs are re-clustered. In this way, nested repeats can be detected and separated from one another. Moreover they will be splitted in 'sub-clusters' according to the presence of long non-conserved regions (i.e. mismatching regions) between instances to deal with interfamily similarity between closely related TE families. The second approach tries to find complete copies of the repeats among all instances of a family. It is searched for the longest sequences in a cluster and filtered according to their occurrence and if it is an active transposon family, there will be at least a few copies found, normally more than three [BQ07].

3.1.2 Homology-based

Homology-based methods use known TEs (of other species) to detect through homology (similarity) new TEs in a given genome. Obviously, homology-based methods are biased towards the detection of previously identified TE families and to TEs active recently enough to retain substantial protein homology.

These methods have strong advantages in finding known transposons and degraded transposons although they fail to recognize transposons unrelated to the known ones. They are also not applicable to certain classes of TEs that are composed entirely of non-coding sequences, such as miniature inverted repeat TEs (MITEs) and short interspersed nuclear elements (SINEs) [KUC+11] [BQ07].

One common approach is to align a genome using fast alignment algorithms with known TEs and analyze the results of the hits. Another known approach uses Hidden Markov Models to detect common TEs. This second approach, despite being effective for closely related genomes, fails when used in distantly related species.

3.1.3 Structure-based

Structure-based methods use prior knowledge about the common structural features shared by different TEs that are necessary for the process of transposition, such as long terminal repeats. These methods rely on detecting specific models of TE architecture, rather than just the expected results of the transposition process, making them less sensitive to similarity between the sequences and the known transposons. Like homology-based methods, these can also detect low copy number families of transposons. Purely structure-based methods are limited by the fact that specific models must be designed and implemented for each type of transposon in analysis and that some classes of transposons have stronger structure characteristics than others and, therefore, are more easily detected using these kinds of methods [BQ07].

Several structure-based methods have been developed recently to detect LTR retrotransposons, by searching for the common structural signals in this subclass of LTR TEs, target site duplications (TSDs), primer-binding sites (PBSs), polypurine tracts (PPTs) and ORFs for the GAG, POL (containing the RT domain) and/or env genes (viral proteins) [XW07] [HW10].

3.1.4 Comparative Genomics

Firstly described by Caspi and Pachter [CP06], comparative genomics methods use multiple alignments of closely related genomes to detect large changes between genomes. The idea behind this method is that insertion regions can be TEs or caused by TEs.

These methods are based on the fact that transposition creates large insertions that can be detected in multiple sequence alignments [BQ07]. They start by searching for insertions regions in whole-genome multiple alignments where there are disrupted sequences by large insertions, normally more than 200 bp. After applying filtering and concatenating the insertion regions are locally aligned with all other insertion regions to identify repeat insertion regions.

This approach can identify new transposon families and instances although it is dependent on the quality of whole genome alignments, which can be compounded by the multiple alignment of draft genomes.

3.2 Transposon Detection Tools

In this section the software tools that will be used on this work are described and grouped accordingly to Bergman and Quesneville [BQ07] previously described.

3.2.1 *De novo* Tools

Table 3.2 shows *De novo* tools that we will consider in our analysis work.

Name	**Description**
PILER[EM05]	A package of efficient search algorithms for identifying characteristic patterns of local alignments induced by certain classes of repeats.

Table 3.2: *De novo* Tools

3.2.2 Homology-Based Tools

Table 3.3 shows the tools that we will use in our study.

Name	**Description**
BLAT[Ken02]	A software tool which performs rapid mRNA/DNA and cross-species protein alignments.
CENSOR[JKDP96]	A software tool designed to identify and eliminate fragments of DNA sequences homologous to any chosen reference sequences, in particular to repetitivte elements.
RepeatMasker[Smi]	A program that screens DNA sequences for interspersed repeats and low complexity DNA sequences. The output of the program is a detailed annotation of the repeats that are present in the query sequence as well as a modified version of the query sequence in which all the annotated repeats have been masked.

Table 3.3: Homology-Based Tools

3.2.3 Structure-Based Tools

Table 3.4 shows the tools that we will use in our study.

Name	**Description**
LTR_Finder[XW07]	A program that predict locations and structure of full-length LTR retrotransposons accurately by considering common structural features.

Table 3.4: Structure-Based Tools

Starting with *de novo* tools, PILER adopts a novel heuristic-based approach to *de novo* repeat annotation that exploits characteristic patterns of local alignments induced by certain classes of repeats. The PILER algorithm is designed to analyze assembled genomic regions and find only repeat families whose structure is characteristic of known subclasses of repetitive sequences. PILER works on the premise that the entire DNA sequence is assembled with a reasonably low number of errors because the algorithm is completely dependent on the position of repeats in the genome for all classification. The output of the clustering step is recorded in terms of start and end coordinates. Similar elements are then clustered into piles, which are sets of overlapping hits [EM05].

Regarding the homology-based tools, BLAT is a mRNA/DNA alignment tool. It uses an index of all non-overlapping K-mers in a given genome to find regions in the genome likely to be homologous to the query sequence. It performs an alignment between homologous regions and stitches together these aligned regions into larger alignments. Finally, BLAT searches for small internal exons possibly missed at the first stage and adjusts large gap boundaries that have canonical splice sites where feasible [Ken02].

CENSOR is a program designed to identify and eliminate fragments of DNA sequences homologous to any chosen reference sequences. It uses BLAST to identify matches between input sequences and a reference library of known repetitive sequences. The length and number of gaps in both the query and library sequences are considered along with the length of the alignment in generating similarity scores. This tool reports the positions of the matching regions of the query sequence along with their classification [SBMP08, JKDP96].

RepeatMasker discovers repeats and removes them to prevent complications in downstream analysissequence assembly and gene characterization. This tool includes a set of statistically optimal scoring matrices permitting estimation of the divergence of query sequences compared to a curated repeat library. A search engine such as BLAST, WU-BLAST, or Crossmatch is utilized in the comparison process. The degree of similarity required for a query sequence to be paired with a reference sequence can be specified by the user. Identification of repeats by RepeatMasker is based entirely upon shared similarity between library repeat sequences and query sequences. The output of the program is a detailed annotation of the repeats that are present in the query sequence as well as a modified version of the query sequence in which all the annotated repeats have been masked [Smi].

In the structure-based tools we have LTR_Finder. Given DNA sequences, it predicts locations and structure of full-length LTR retrotransposons accurately by considering common structural features. LTR_FINDER identifies full-length LTR element models in genomic sequence in four main steps. It first selects possible LTR pairs by searching for all exactly matched string pairs in the input sequence by a linear time suffix-array algorithm. Then it selects pairs whose distances and overall sizes satisfy a set of given restrictions. It calculates the distances and then records the LTR candidates for further analysis. After that, Smith–Waterman algorithm is used to adjust the near-end regions of LTR candidates to get alignment boundaries. At the end of this step, a set of regions in the input sequence is marked as possible loci for further verification. Secondly, LTR_FINDER tries to find signals in near LTR regions inside these loci. The program detects PBS by aligning these regions to the 30tail of tRNAs and PPT by counting purines in a 15 bp sliding window along these regions. This step produces reliable candidates. In the end, this program reports possible LTR retrotransposon models at different confidence levels [XW07].

Chapter 4

Empirical Evaluation of TE detection tools

This chapter presents the experimental setup and the empirical evaluation of TE detection tools.

In Section 4.1 the dataset generation process and resources are detailed. The Section 4.2 it is established how the different transposon detection tools are going to be evaluated and the expected outcomes of this evaluation. The results of the assessment of each tools are detailed. In Section 4.2.7 the results of the different tools' assessment are compared. Finally in Section 4.3 we present the chapter's conclusions.

4.1 Experimental Setup

4.1.1 Datasets Generation

In order to evaluate the TE detection tools it is essential to have a curated dataset of genome sequences. With this curated dataset we can signal the errors made by the detection tools.

The datasets will be composed by genome sequences in FASTA formatA with the annotation of the elements in the sequence (TEs, genes or repeatitive elements). The datasets will be as diversified as possible, having distinct genome sequences in order to make as complete as possible evaluation of these tools.

To this end we will propose and implement a Dataset Simulator as described in 4.1.2. It allows the generation of different genome sequences, varying both in length and composition. These sequences will have transposons in different proportions and variety. In addition, these sequences will have genes and other repetitive elements that are not transposons. Finally these sequences can have mutations, either point mutations or *indel* mutations. This last condition makes the dataset more in agreement with what real genomes are like.

4.1.2 Simulator

The simulator produces sequences that have a random number of TEs in random but known positions. The simulation parameters are:

- Sequence Length: length of the output sequence.
- Gene Percentage: % of genes that should be included in the sequence in relation to its total length.
- Transposon Percentage: % of transposons that should be included in the sequence in relation to its total length.
- Repetitive Elements Percentage: % of repetitive elements (no transposons included) that should be included in the sequence in relation to its total length.
- Set of Genes: a set of genes in FASTA format that are used in the creation of this sequence.
- Set of Transposable elements: a set of TEs in FASTA format that are used in the creation of this sequence.
- Set of Repetitive elements: a set of repetitive elements (no transposons included) in FASTA format that are used in the creation of this sequence.
- Rate of Insertions: number of inserted nucleotides per 1000 bp.
- Rate of Deletions: number of deleted nucleotides per 1000 bp.
- Rate of Replacements: number of substituted nucleotides per 1000 bp.

The output of the simulator is a set of sequences, written in FASTA format, and an annotation file containing all TEs and repetitive elements locations inside each sequence.

The resulting sequence consists of genes, transposons and other repetitive elements filled with random nucleotides in the gaps between them. The quantity of TEs, genes and repetitive elements are defined by the parameters Transposon, Gene and Repetitive Elements Percentages respectively. These elements are from the Set of Genes, TEs and Repetitive Elements given in the input parameters.

There is also the possibility for the generated sequence to have point mutations using the Rate of Replacements parameter, replacing a given number of bases per 1000 bp. In the same way, *indel* mutations [KR04] can also be applied to the result sequence by changing the Rate of Insertions and Deletions parameters.

The sequences generated are personalized according to the situation to be tested. For instance, if the aim of the sequence generated is to evaluate the transposon detection tools regarding LTR transposon detection then the set of TEs given should consist of elements of this class.

The sequence generator was written in Perl using Bioperl [bio11], a publicaly available package of Perl code useful for biology related tasks. The dataset simulator is a *Bash* script that uses the sequence generator to create all the sequences according to pre-defined parameters.

The input values for the dataset generation are presented in Table 4.1.

Name	**Description**
Sequence Length	10000000bp
Transposon Percentage	From 40% to 80% in 10% increment steps.
Gene Percentage	4% or 10%.
Repetitive Elements Percentage	From 16% to 46% in 10% increment steps.
Rate of Insertions, Deletions and Replacements	From 0 to 20 for 1000bp in 10 increment steps.
Set of Transposable elements	Repbase and Gydb TEs, divided in 10 categories: autonomous LTR Retrotransposons, non autonomous LTR Retrotransposons, DIRS, autonomous non-LTR Retrotransposons, non autonomous non-LTR Retrotransposons, TIRs, DNA Transposons, Helitrons and Politrons.
Set of Genes	Repbase genes.
Set of Repetitive elements	Repbase repetitive elements.
Number of Sequences per combination	It was chosen that we will create 50 sequences for each combination of the mentioned inputs.

Table 4.1: Input values for dataset generation

Each sequence of our dataset was named according to the parameter set that was used to generate it. The nomenclature is as follows:

seq *ijmtnnn*

where,

- i: a value from 0 to 4. This value defines the percentage of TEs and repetitive elements in the sequence. Higher value means higher percentage of TEs and lower percentage of repetitive elements.
- j: a value from 0 to 1. This value defines whether gene percentage is 4% or 10%.
- m: a value from 0 to 2. This value indicates the percentage of mutations in the sequence.
- t: a value from 0 to 9. This value indicates which of the ten TEs categories this sequence has in its composition.
- n: a value from 0 to 49. This value indicates the number of the sequence for each combination of parameters.

4.1.2.1 Sequence Generator Algorithm

Given the input parameters defined in Section 4.1.2 the first step is to read all the transposons, genes and repetitive elements files.

For each category of elements, the algorithm searches for all the files in the input folder and collects the identification name and length of each element. Using these collected elements, the algorithm iterates through the collections of TEs, genes and repetitive elements and randomly selects the elements to be present in the output sequence up until the length allocated for them (percentage of elements * Sequence Length) is achieved, with an maximum error of 5%.

The next step in the algorithm is to write a FASTA file with all the different TEs that will be present in the final sequence. This file can later be used to analyze TEs features.

In order to complete the sequence length, it sums all the elements' length and calculates the remain random nucleotides to be added to the output sequence.

Using all the previous info, the algorithm enters in a loop where it randomly chooses to insert either a random nucleotide, a gene, a repetitive element or a TE. If there are mutation rates, mutations are applied and then the element is written in the output sequence and annotated separately in a GFF file with the relevant information, namely the identification name, the start and end positions, the length and the type (in case of a TE).

As for the mutation insertion in the elements, the algorithm uses probability values based on the sample model present in the Evolver Software [Rob]. Using the Rate of Insertions, Rate of Deletions and Rate of Replacements given in the input, it goes through all the nucleotides present in the element and inserts, removes or replaces according to the probability of the rates given and based on the probabilities of insertions, deletions and replacements present in the Evolver model.

At the end of this process the following files are generated:

- Sequence file in FASTA format;
- Annotation file in GFF format, including all the elements positions and types and the sequence's properties;
- A FASTA file with all the transposable elements present in this sequence;
- A file with the name of the TEs in the sequence and the number of each present in the sequence.

4.1.3 Raw Data

The simulator that produces the artificial dataset requires data related to TEs, genes and repetitive elements all in FASTA format.

In this dissertation we use a set of genes from FlyBase [Fly] which are real genes annotated from Drosophila Melanogaster, a fly species.

The sources for TEs were: Repbase [Rep] which is a database of repetitive DNA elements and from Gydb [Gyd] which is also a cooperative repository of TEs organized and classified by type.

4.1.4 Hardware Setup and time performance

The computational power needed for the dataset generation, tools' assessment and machine learning lead us to use a set of four computer nodes with the following specifications.

- Processor: 2x Intel Xeon with 4 cores each, running at 2GHz with 12MB cache L2
- Memory: 32GB of RAM memory

Computation time was also a variable we have taken into account due to the dimension of our datasets. In table 4.2 we present average estimates for the times the different tasks consumed.

Task	**Cores used**	**Average Time (seconds)**
Sequence Generation (one sequence)	1	1000
LTR Finder assessment (one sequence)	1	800
RepeatMasker assessment (one sequence)	4	650
Censor assessment (one sequence)	4	350
PILER assessment (one sequence)	1	200
BLAT asssessment (one sequence)	1	230
Merge BLAT results (one sequence)	1	120
Merge Censor results (one sequence)	1	150
Merge LTR Finder results (one sequence)	1	25
Merge PILER results (one sequence)	1	120
Merge RepeatMasker results (one sequence)	1	150
Generate TE oriented table (one sequence)	1	1200
Generate sequence oriented table (one sequence)	1	350

Table 4.2: Time performance table

4.2 Evaluation of Transposon Detection Tools

Evaluating the different transposon detection tools is a key step in the scope of this dissertation. Using a dataset generated with the simulator described in Section 4.1.2, each transposon detection tool was used to process the dataset and to predict the TEs.

Each tool analyzes all the dataset sequences and produced as a result the annotations of the TEs. These annotations were then compared with the annotations generated by the simulator and the general accuracy was measured for that tool on the detection of TEs. Furthermore it enabled a more in-depth analysis regarding the tool's capacity in detecting specific sequences generated by the simulator.

4.2.1 Data Organization

Using the gathered data, we organized our results in two distinct data tables. The Table 4.3 is oriented to each dataset sequence. It consists of a file in which each line describes how a tool performs on a given dataset sequence.

Parameter	**Description**
Tool	Tool name.
Seq	Sequence's name.
% TE	TEs' percentage in this sequence.
% Gene	Genes' percentage in this sequence.
% Repetitive Element	Repetitive elements' percentage in this sequence.
% Mutation	Mutation percentage present in this sequence.
Autonomous LTR retrotransposons	Number of autonomous LTR retrotransposons in this sequence.
Non autonomous LTR retrotransposons	Number of non autonomous LTR retrotransposons in this sequence.
DIRS	Number of DIRS in this sequence.
Non-LTR retrotransposons	Number of non-LTR retrotransposons in this sequence.
Autonomous non-LTR retrotransposons	Number of autonomous non-LTR retrotransposons in this sequence.
Non autonomous non-LTR retrotransposons	Number of non autonomous non-LTR retrotransposons in this sequence.
TIRs	Number of TIRs in this sequence.
DNA transposons	Number of DNA transposons in this sequence.
Helitrons	Number of Helitrons in this sequence.
Polintons	Number of Polintons in this sequence.
Average Score	Average score of the tool in identifying TEs in this sequence.
TE found	Number of TEs candidates that the tool found in this sequence.
Correct TE found	Number of correct TEs identified by the tool in this sequence.

Table 4.3: Sequence oriented table

On a second table (Table 4.4) we grouped the information based on a TE perspective. Each line describes a TE in a sequence and the results of each tool in descovering that particular TE.

Parameter	**Description**
Seq	Sequence's name.
TE type	The TE type.
Start	The position where it appears in the sequence.
End	End position of the TE in the sequence.
Length	TE's length.
BLAT start	Error of the start position of BLAT detection (0 if no error and no value if TE is not found).
BLAT end	Error of the end position of BLAT detection (0 if no error and no value if TE is not found).
BLAT length	Error of the length of the TE comparing to BLAT detection (0 if no error and no value if TE is not found).
LTR Finder start	Error of the start position of LTR Finder detection (0 if no error and no value if TE is not found).
LTR Finder end	Error of the end position of LTR Finder detection (0 if no error and no value if TE is not found).
LTR Finder length	Error of the length of the TE comparing to LTR Finder detection (0 if no error and no value if TE is not found).
Censor start	Error of the start position of Censor detection (0 if no error and no value if TE is not found).
Censor end	Error of the start position of Censor detection (0 if no error and no value if TE is not found).
Censor length	Error of the length of the TE comparing to Censor detection (0 if no error and no value if TE is not found).
Piler start	Error of the start position of Piler detection (0 if no error and no value if TE is not found).
Piler end	Error of the start position of Piler detection (0 if no error and no value if TE is not found).
Piler length	Error of the length of the TE comparing to Piler detection (0 if no error and no value if TE is not found).
RepeatMasker start	Error of the start position of RepeatMasker detection (0 if no error and no value if TE is not found).
RepeatMasker end	Error of the start position of RepeatMasker detection (0 if no error and no value if TE is not found).
RepeatMasker length	Error of the length of the TE comparing to RepeatMasker detection (0 if no error and no value if TE is not found).

Table 4.4: TE oriented table

In the next subsections we describ the obtained results of each tools. We first analyze the results of each tool separately and then we make an overall comparison on the results to get the global conclusions.

4.2.2 BLAT Results

The results of BLAT assessment is shown in Figure 4.1. It shows the correctness of BLAT as a function of all the combination of different input parameters such as TE, genes, repetitive element and mutations percentages. This representation suggests a strong evidence that the mutation percentage has a negative effect regarding the TE detection capabilities of this tool.

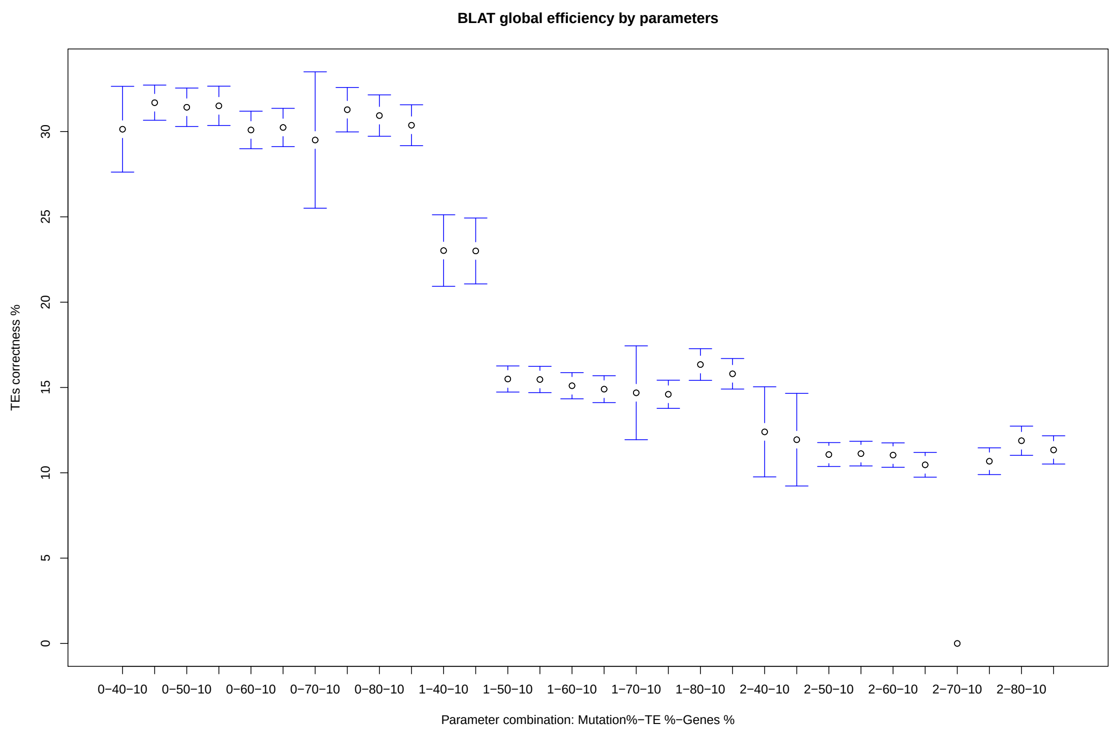

Figure 4.1: BLAT's accuracy plotted against the parameters combination values

In Figure 4.2 it is shown the results of the assessment of BLAT in each of the ten categories of TEs.

The results show that this tools' performance is between 18% and 28% in almost all TE categories. The exceptions are DNA TEs and Polinton TEs which the tool identifies in less than 18% of the cases and DIR TEs which have a discover rate bellow 5%. On the other hand, this tools' performance regarding discovery of non autonomous non LTR retrotransposons is better than average as it reaches above 35% of efficiency.

In Figures 4.3 and 4.4 the internal and overall correctness of this tool in finding TE is shown.

Figure 4.3 shows that BLAT's internal efficiency varies between 0 to 60% of correct TE founds per candidates, being the 5% to 10% the most frequent occurrence.

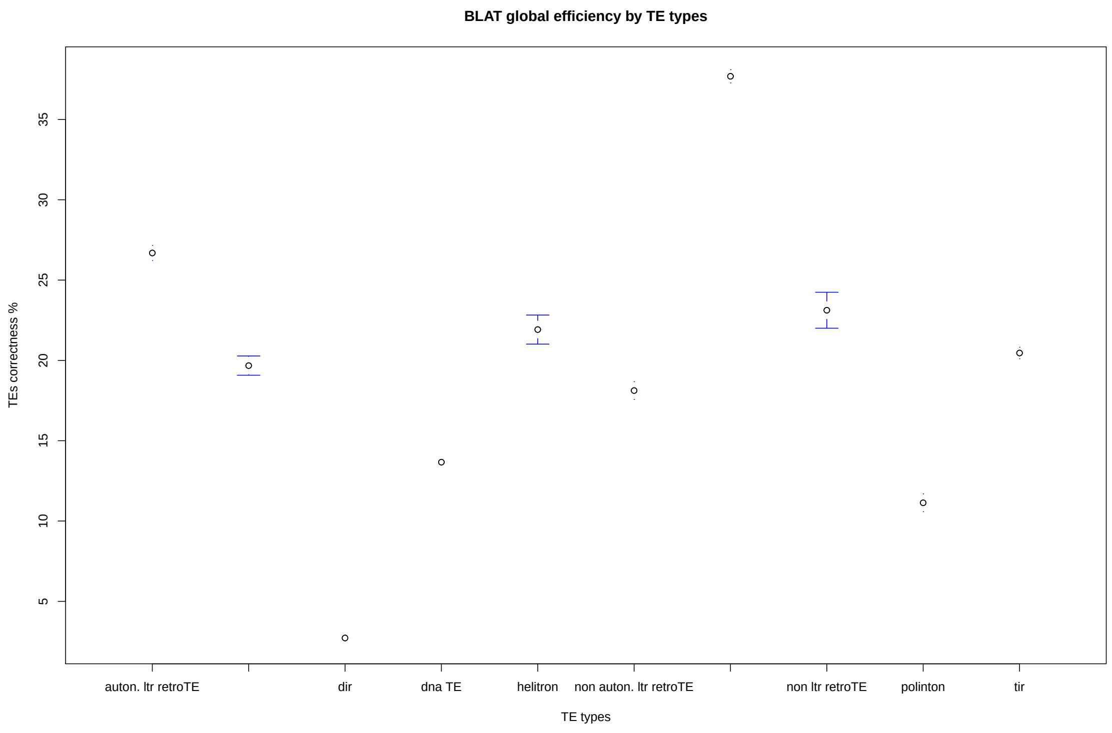

Figure 4.2: BLAT's accuracy according to TE types

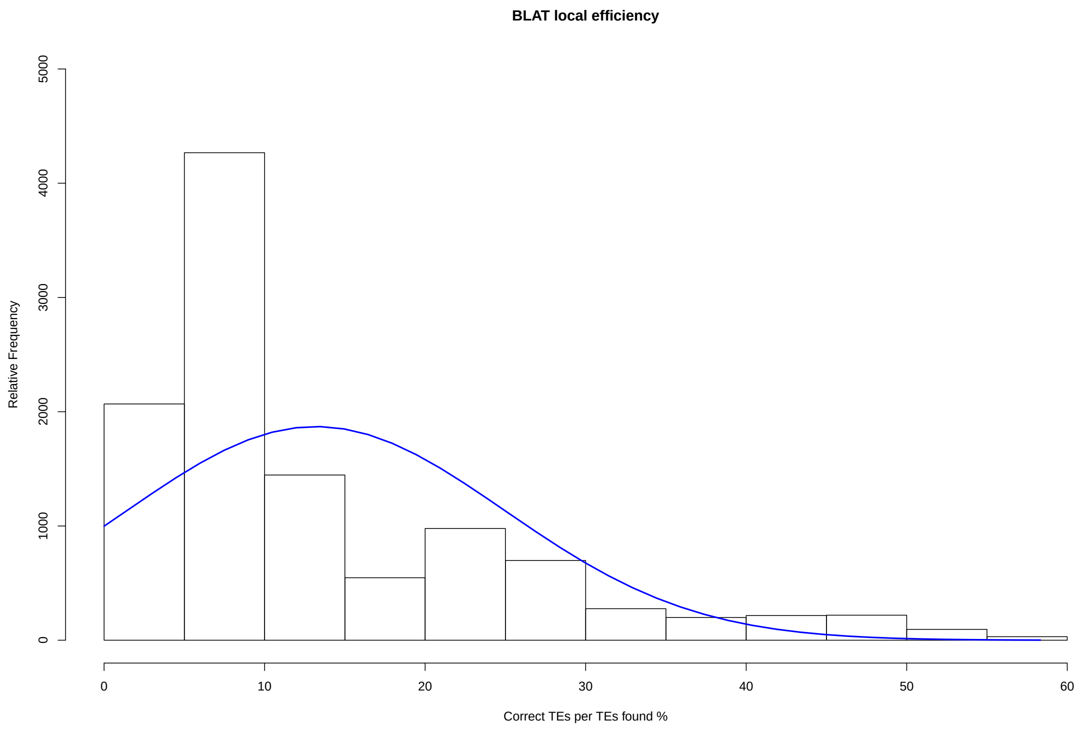

Figure 4.3: BLAT's intern accuracy

As it can be seen in Figure 4.4, the global efficiency of this tool varies from 0% to roughly 58% with the the most frequent evaluations lower than 20%.

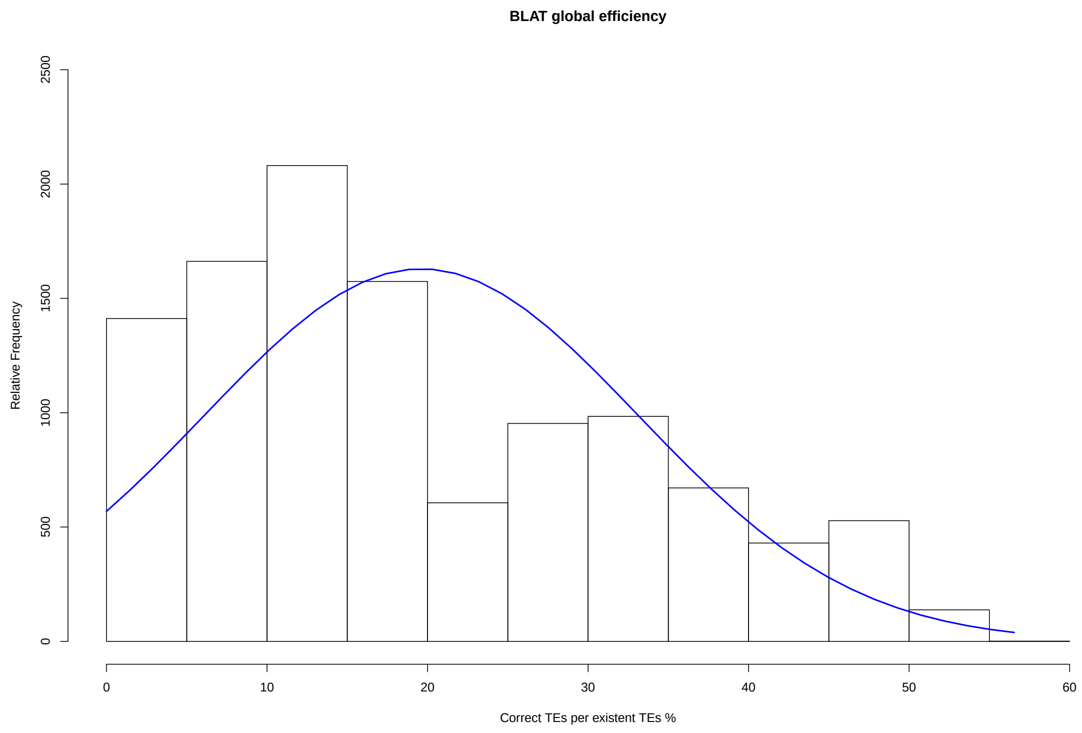

Figure 4.4: BLAT's actual accuracy

To better understand the importance of the mutation variable in TE discovery in these sequences, we show, in Figure 4.5, how BLAT's correctness is influenced by the mutation percentage.

As Figure 4.5 shows when the mutation percentage increases, the efficiency of this tool decreases monotonicaly. The correlation coefficient between mutation percentage and the number of correct TEs found by BLAT is -0.6. Therefore we can assume that mutation percentage has impact on BLAT's efficiency.

The relevance of gene percentage in a sequence is presented in Figure 4.6.

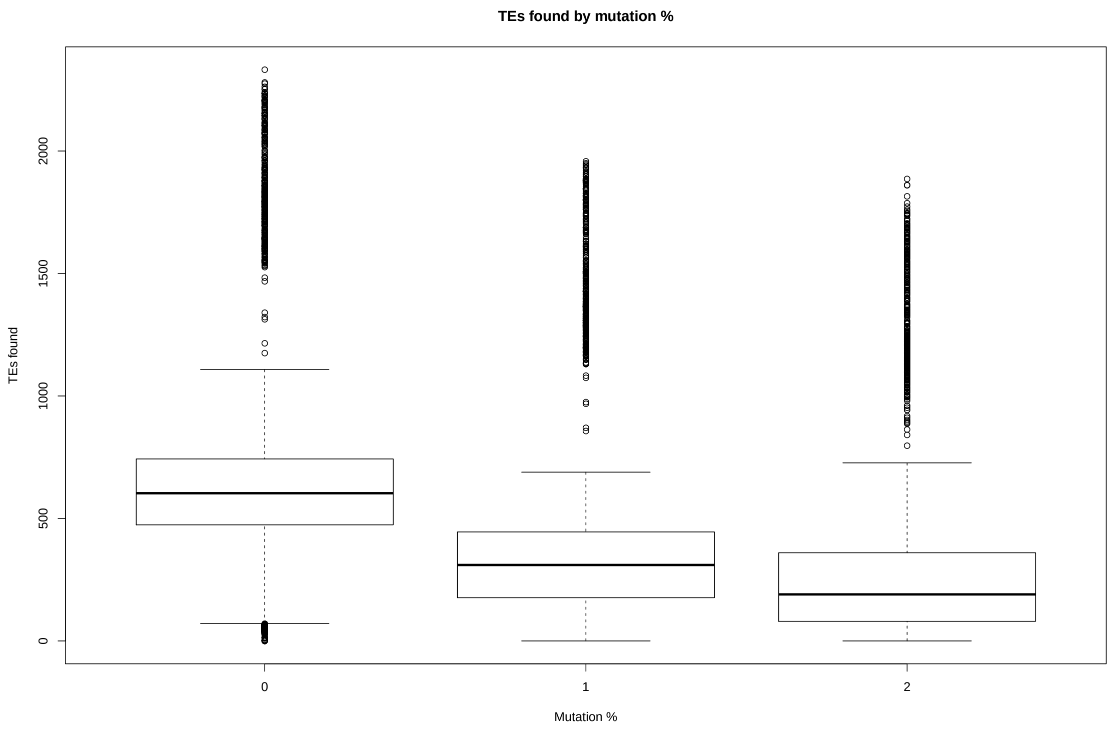

Figure 4.5: Corrrect TEs found per mutation percentage

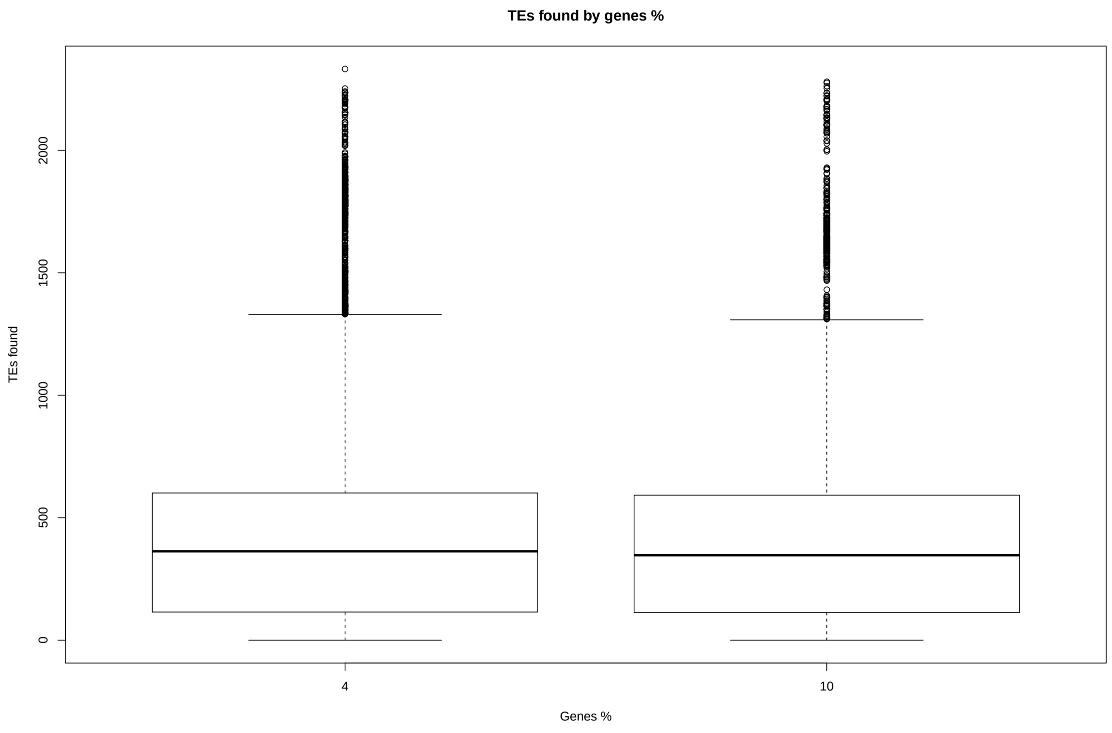

Figure 4.6: Corrrect TEs found according to gene

There is no statistical evidence that gene percentage is related with the tool's performance. This is supported by the correlation coefficient value of -0.01 that demonstrate that these two variables are not linearly related.

We can analyze the influence of the TE and repetitive elements percentage in each sequence on the BLAT's performance in Figure 4.7.

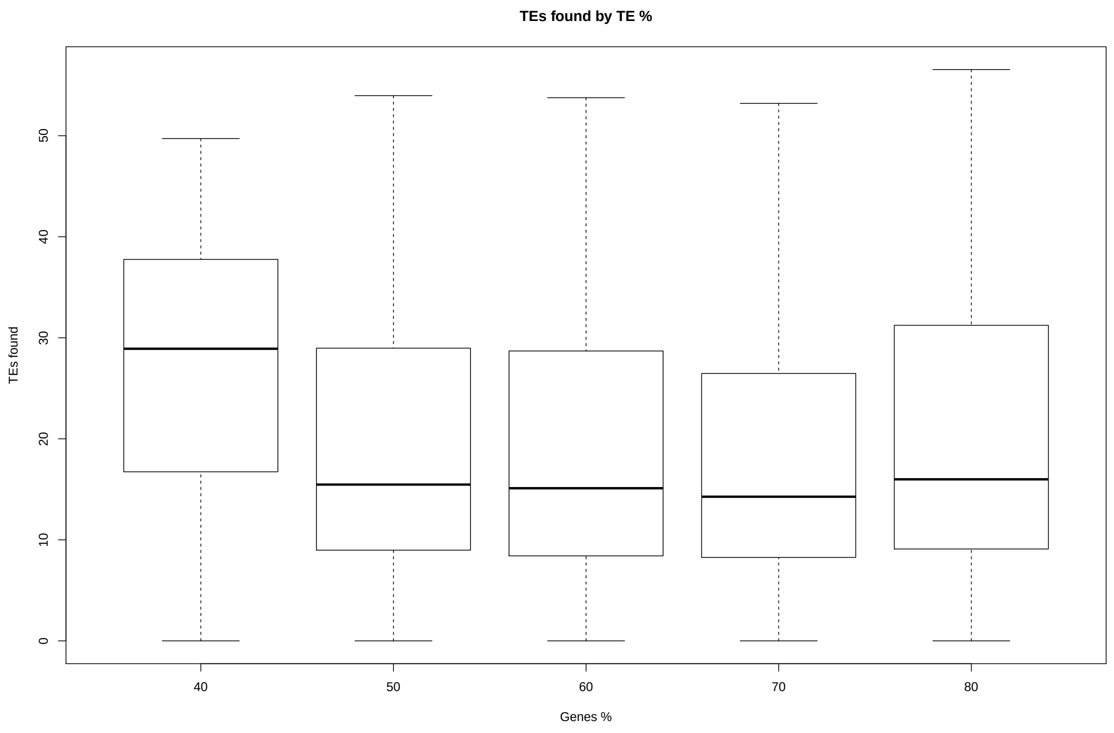

Figure 4.7: Corrrect TEs found according to TE and repetitive elements percentage

Like the case of gene percentage, there is no statistical evidence that the TE and repetitive elements percentage is related with the tool's performance. This is supported by the correlation coefficient value of -0.08.

Analyzing the actual detection error is yet another problem that TE detection tools face. Regarding this, we show in Figures 4.8, 4.9 and 4.10 how well the BLAT tool performs.

As Figure 4.8 shows, the error of BLAT tool is close to inexistent in every TE category with the exception of Polinton TEs. In this case the error is about 12bp in average. In terms of "end error" show in Figure 4.9, it can be ssen that the average error varies from 0 to 0.6. One more time, this tool is not particularly accurate regarding Polinton detection since the standard deviation of this error is higher than the other TEs. Figure 4.10 also shows that the error is practically null in all TE categories except Polinton TEs.

Figure 4.11 shows the average error in TE length detection according to the actual TE length. We can see that if the TE length increases, the standard deviation also tends to increase. Despite this, there is no statistical evidence that the TE length is related with the tool's performance in detecting the actual length. This is supported by the correlation coefficient value of -0.09.

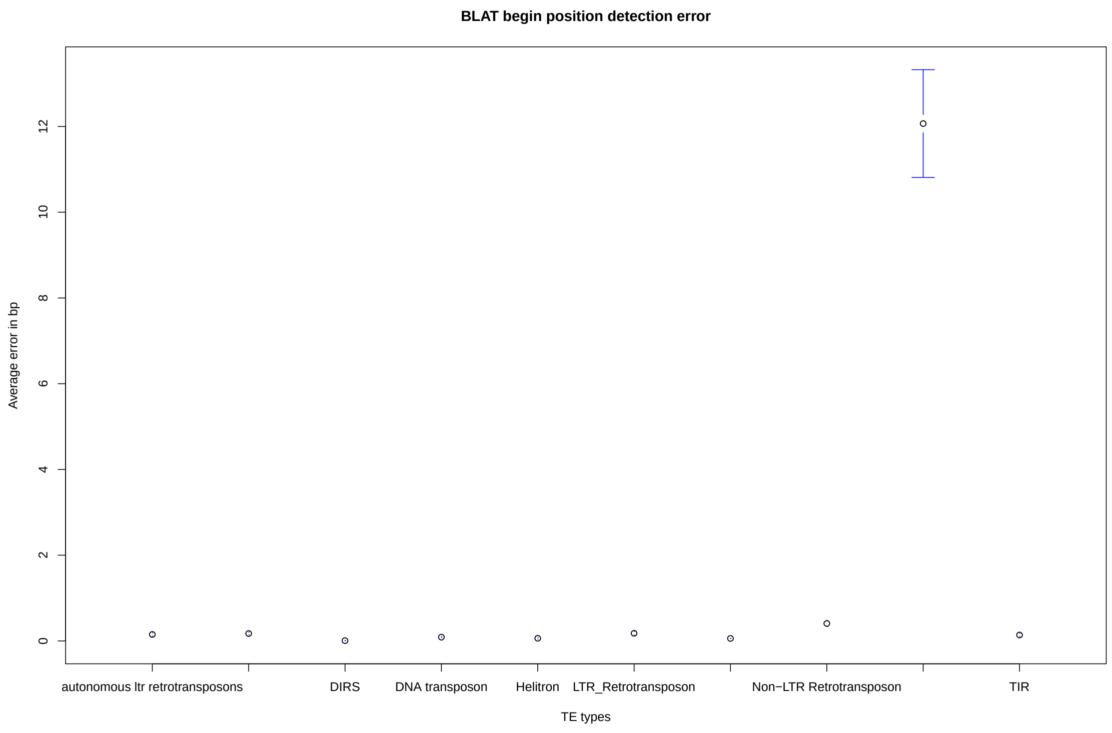

Figure 4.8: BLAT "begin error"

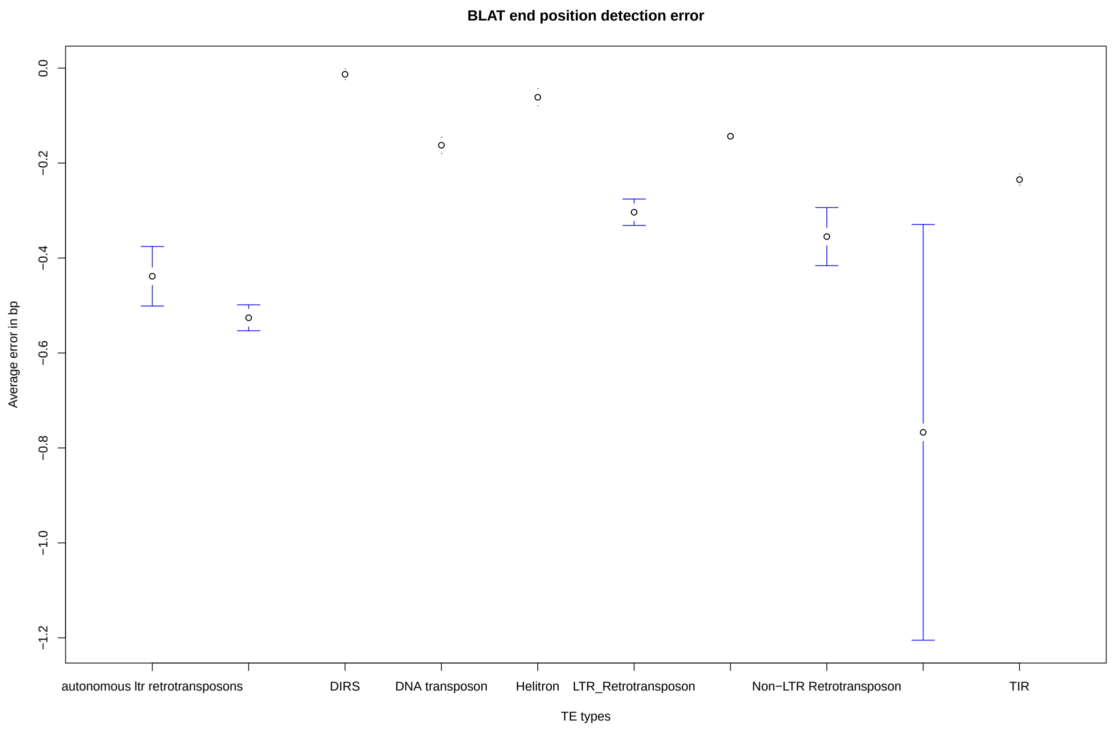

Figure 4.9: BLAT "end error"

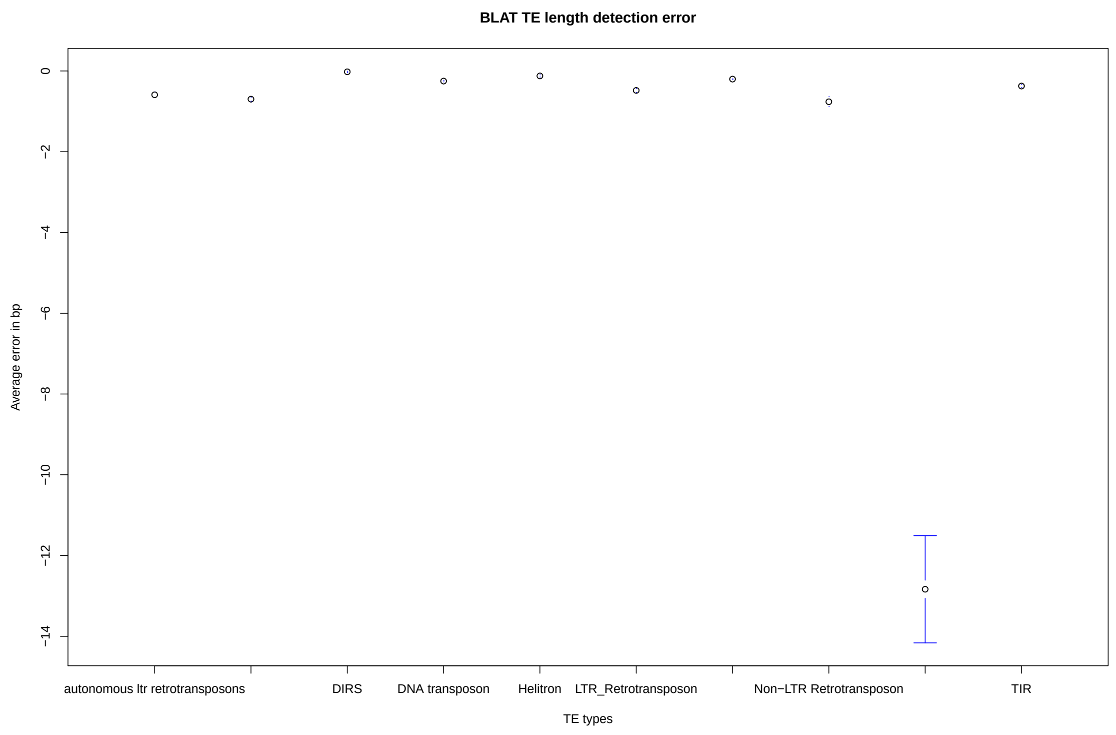

Figure 4.10: BLAT TE "length error"

Figure 4.12 shows the average error in TE length detection according to the mutation percentage of the sequence analyzed. We can see that as the mutation percentage increases so does the average error of TE detection. Nonetheless, there is no statistical evidence that the TE length is related with the tool's performance in detecting the actual length. This is supported by the correlation coefficient value of -0.03.

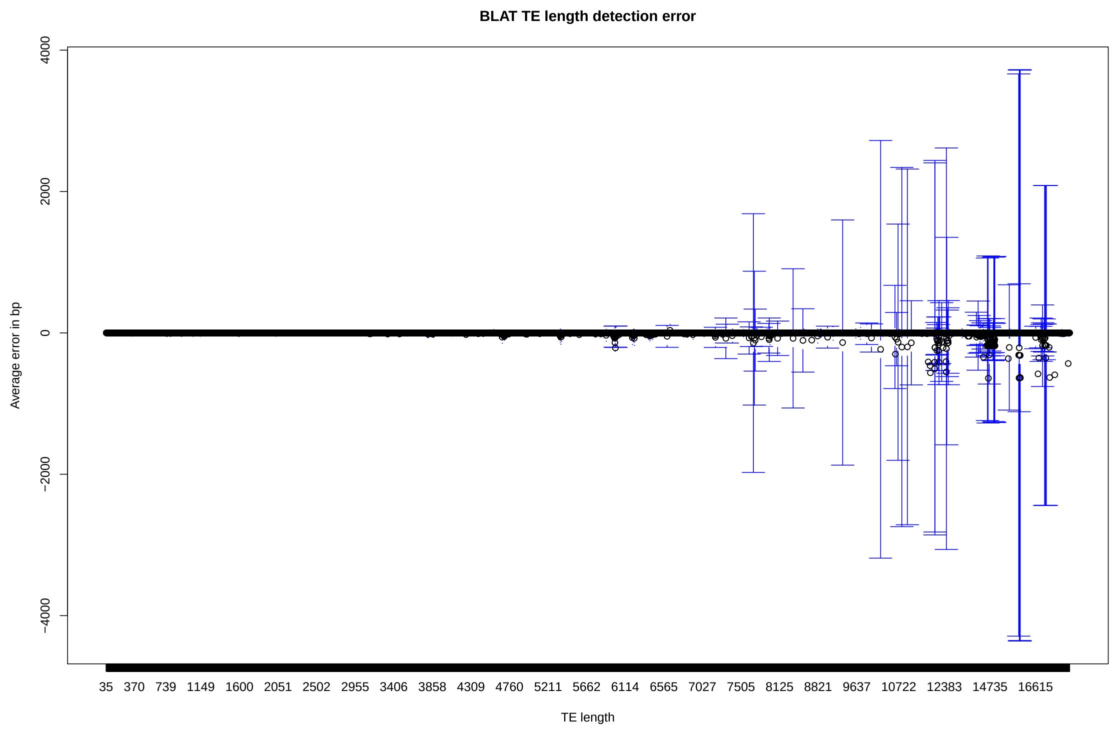

Figure 4.11: BLAT's TE length error

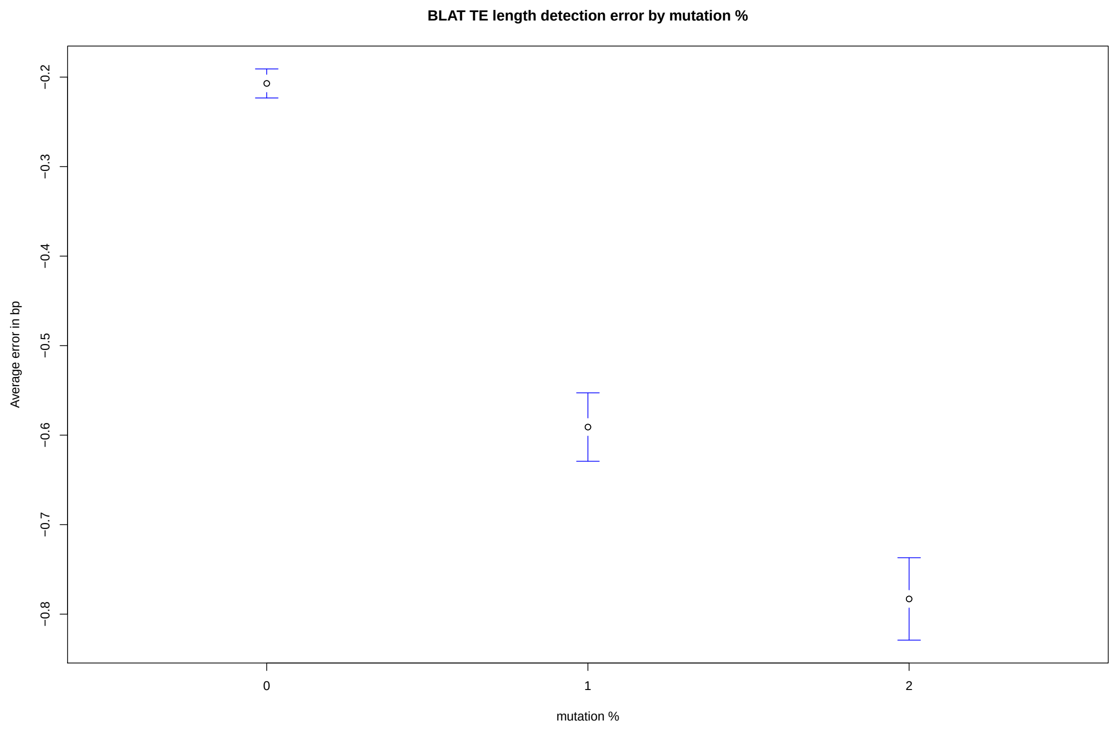

Figure 4.12: BLAT's TE length error

4.2.3 Censor

The results of Censor assessment is shown in Figure 4.13. It shows the correctness of Censor as a function of all the combination of different input parameters such as TE, genes, repetitive element and mutations percentages.

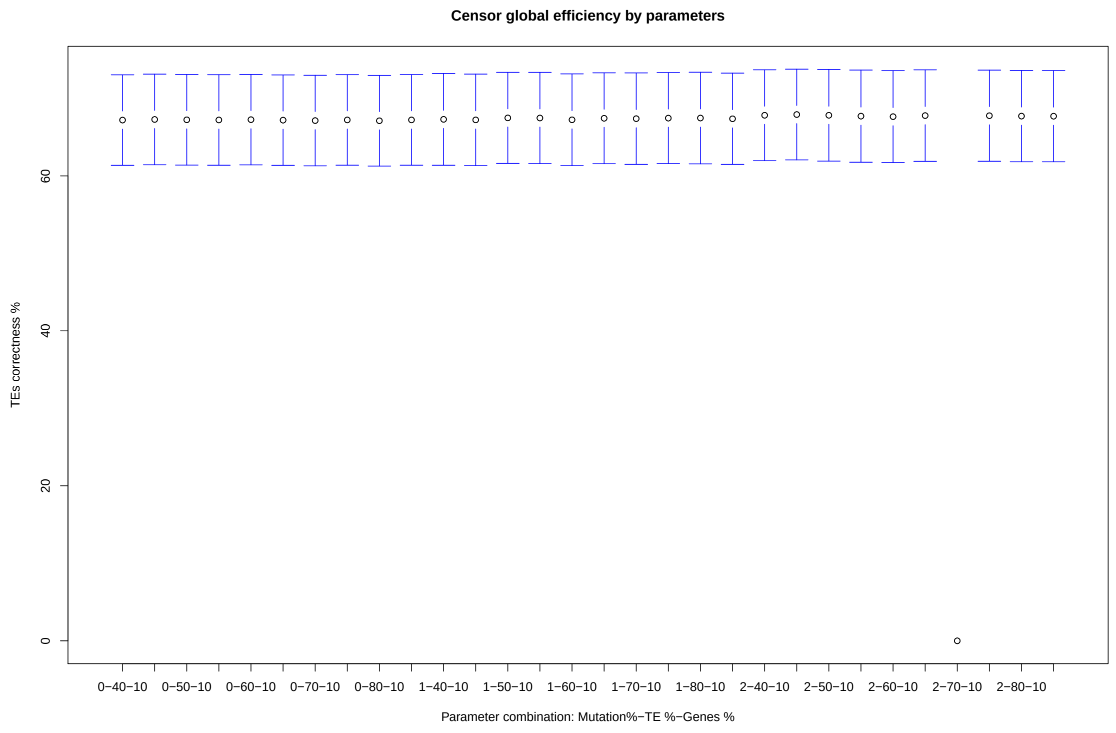

Figure 4.13: Censor's accuracy plotted against the parameters combination values

In Figure 4.14 it is shown the assessment of Censor in each of the ten categories of TEs.

The performance regarding TE detection for all but Polinton TEs is relatively good as it ranges from more than 50% to over than 80% in DIRs, DNA transposons and non autonomous LTR retrotransposons. Polinton detection using this tool is much weaker than with the other classes as the correctness of this tool in this TE category is roughly above 20%.

In Figures 4.15 and 4.16 the internal and overall correctness of this tool in finding TE is shown.

Figure 4.15 shows that Censor's internal efficiency varies between 0% to roughly 50% of correct TE founds per candidates, being the 35% the most frequent occurrence.

As it can be seen in Figure 4.16, the global efficiency of this tool varies from 0% to roughly 82% with the the most frequent evaluations higher than 60%.

To better understand the importance of the mutation variable in TE discovery in these sequences, we show in Figure 4.17 how Censor's correctness is influenced by the mutation percentage.

As Figure 4.17 shows, there is minor to none differences in Censor's detection capabilities when analyzing sequences with 0, 1 or 2 percent of *indel* and point mutations. The correlation coefficient between mutation percentage and the number of correct TEs found by LTR Finder is

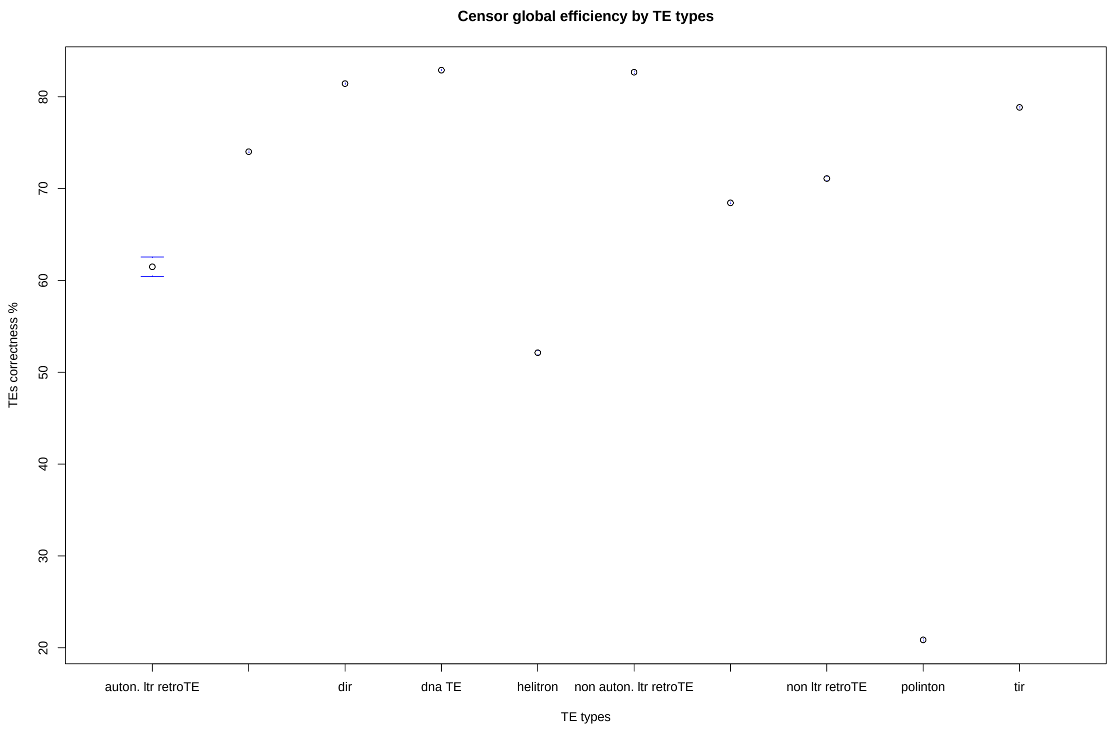

Figure 4.14: Censor's accuracy according TE types

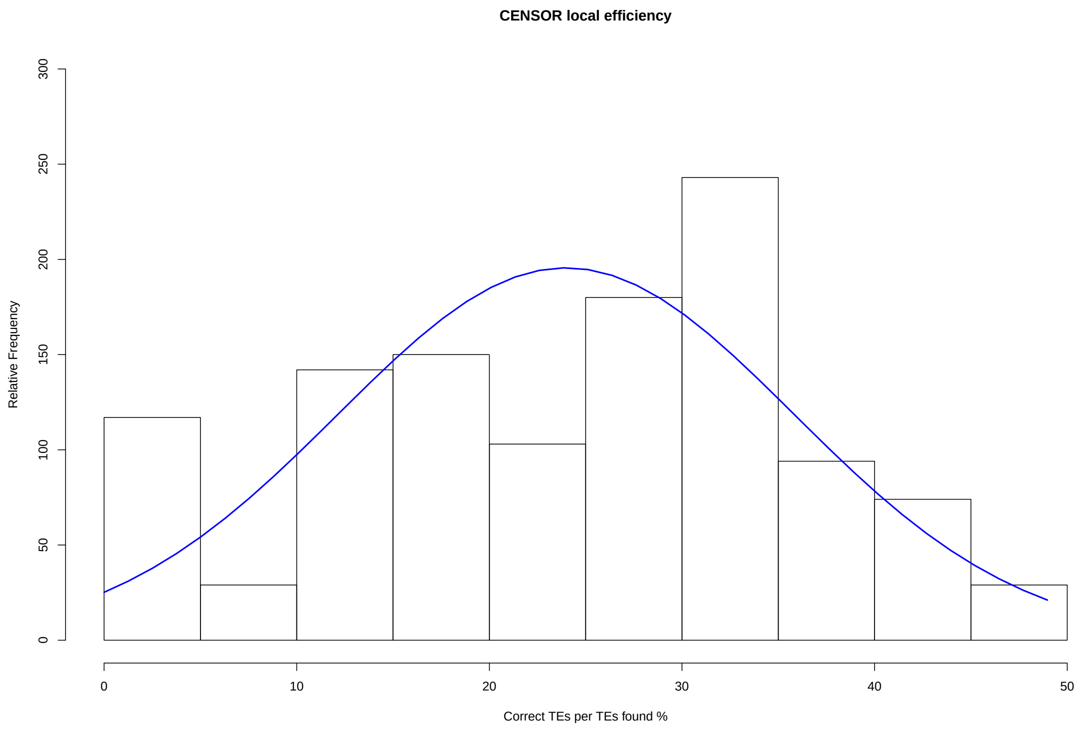

Figure 4.15: Censor's internal accuracy

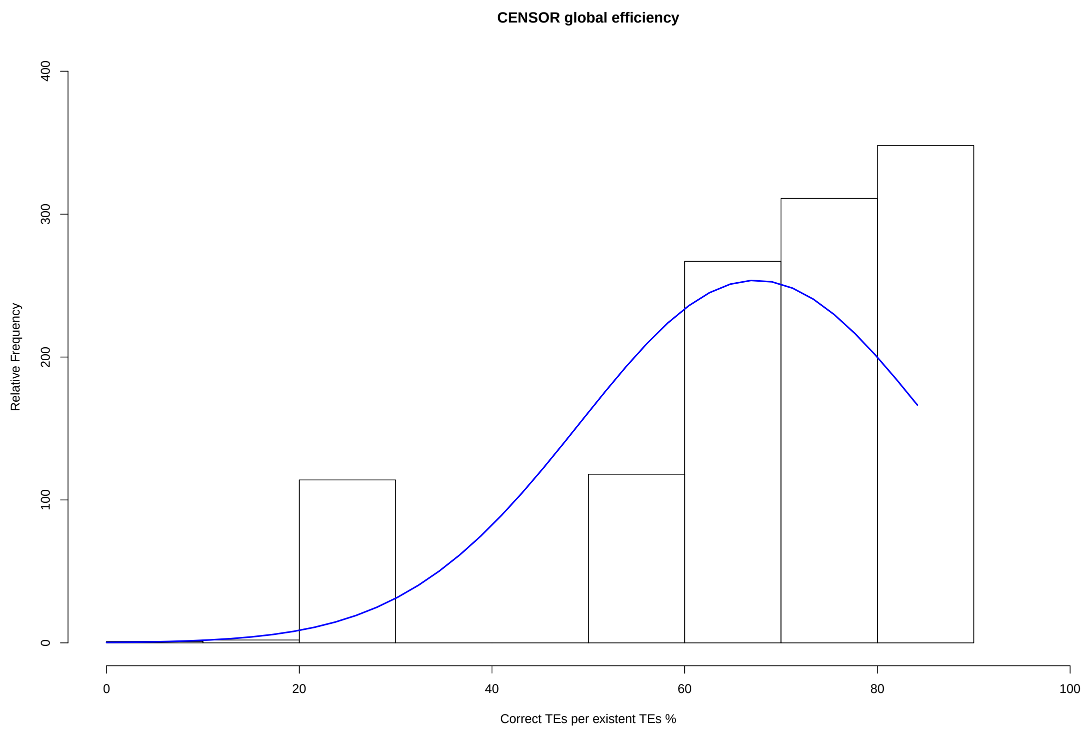

Figure 4.16: Censor's actual accuracy

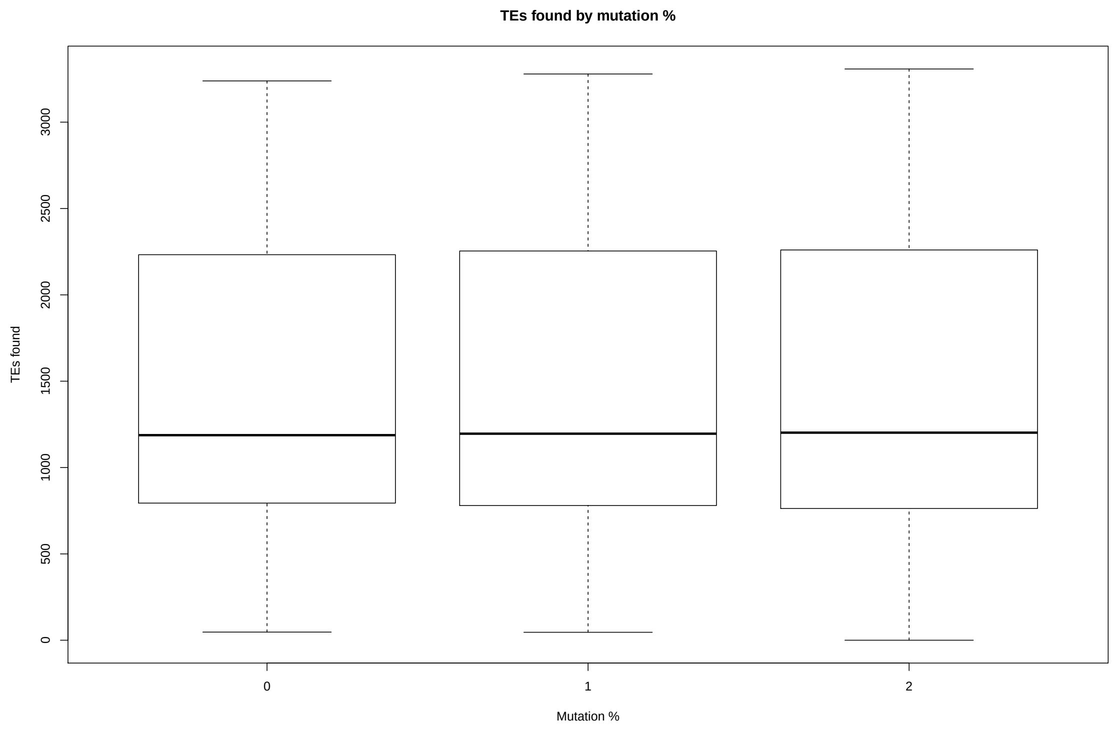

Figure 4.17: Corrrect TEs found per mutation percentage

0.01. Therefore we can assume there is no statistical evidence that mutation percentage is related to Censor's accuracy.

The relevance of gene percentage in a sequence is presented in Figure 4.18.

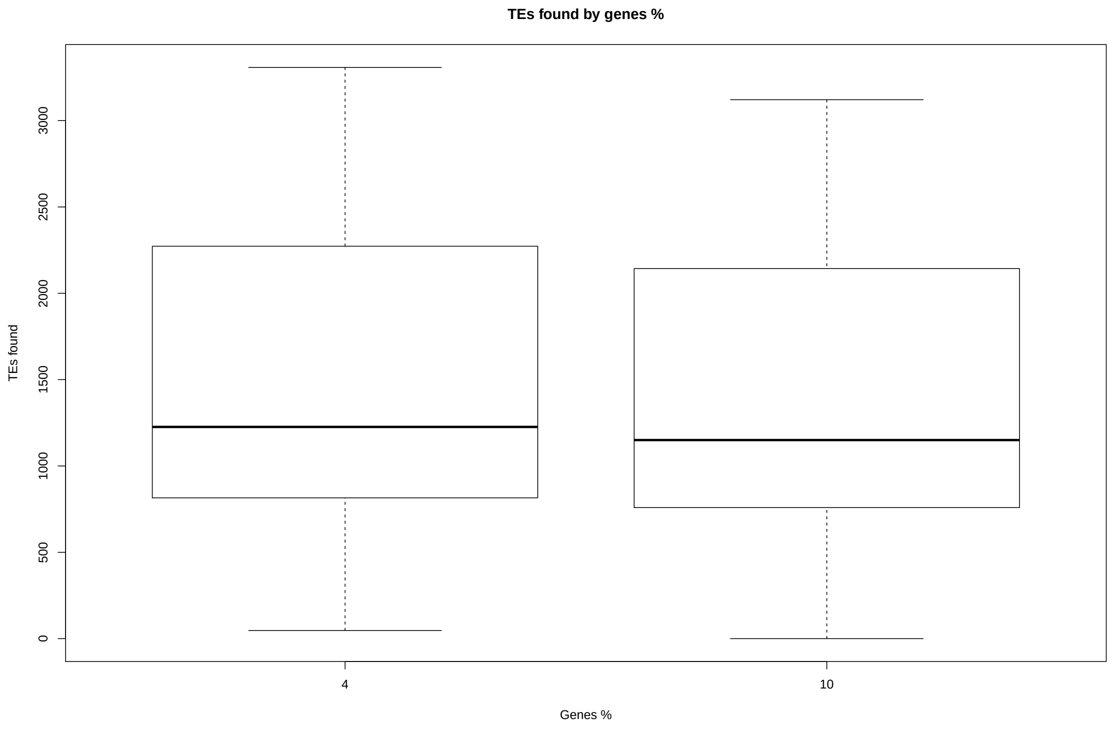

Figure 4.18: Corrrect TEs found according to gene percentage

There is no statistical evidence that gene percentage is related with the tool's performance. This is supported by the correlation coefficient value of 0 that demonstrate that these two variables are not linearly related.

Regarding the influence of the TE and repetitive elements percentage in each sequence on the Censor's performance, in Figure 4.19 it is show how it varies with the TE percentage.

As in the cases of gene and mutations percentage, there is no statistical evidence that the TE and repetitive elements percentage is related with the tool's performance. This is supported by the correlation coefficient value of 0.

Analyzing the actual detection error is yet another problem that TE detection tools face. Regarding this, we show in Figures 4.20, 4.21 and 4.22 how well the Censor tool performs.

As Figure 4.20 shows, the begin position error of Censor tool ranges between 0bp and 2bp in every TE category with the exception of Polinton TEs. In this case the error is about 8bp in average. In terms of "end error", shown on Figure 4.21, it can be seen that the average error varies from 0bp to 2bp. One more time, this tools is not particularly accurate regarding Polinton detection. Also, non-LTR retrotransposons have an higher end error than the rest of TE categories. Figure 4.22 shows that the error in length varies from 0bp to 3bp in all TE categories except non-LTR retrotransposons and Polintons.

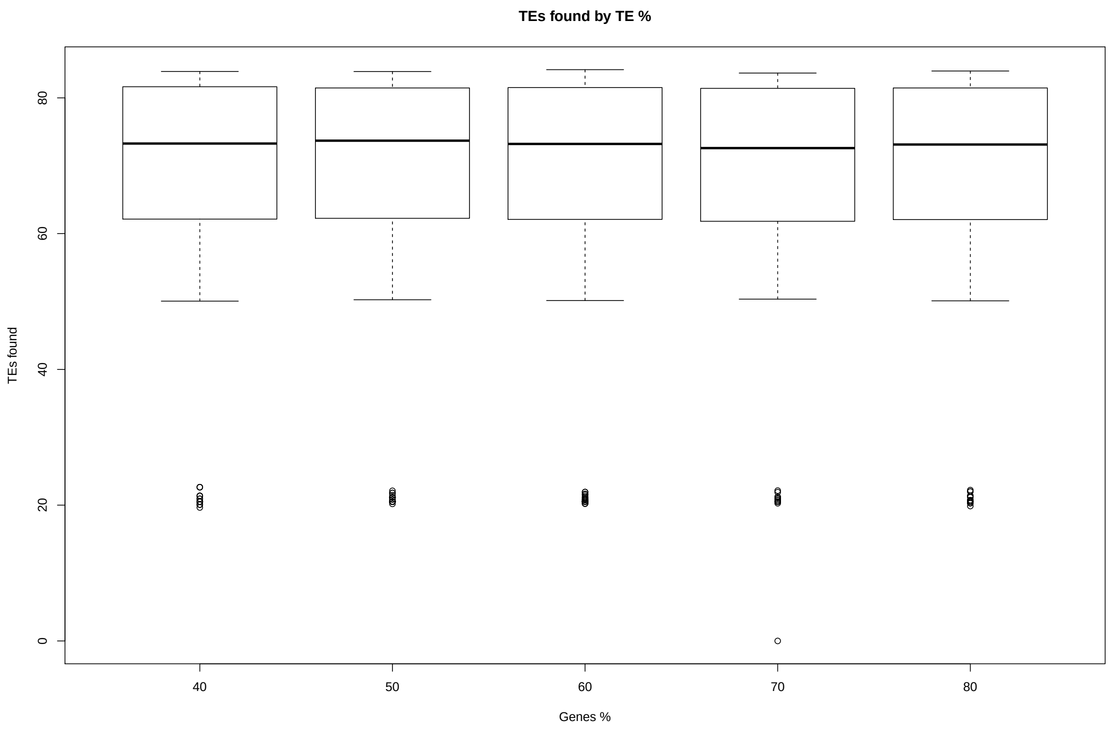

Figure 4.19: Corrrect TEs found according to TE and repetitive elements percentage

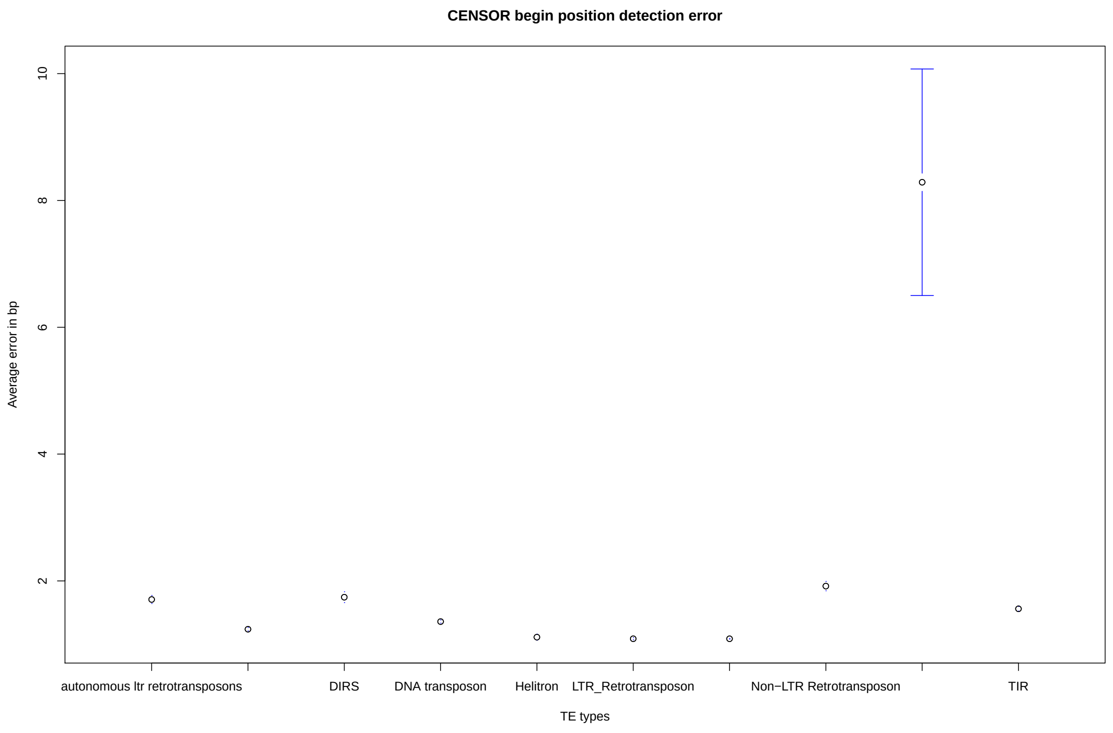

Figure 4.20: Censor "begin error"

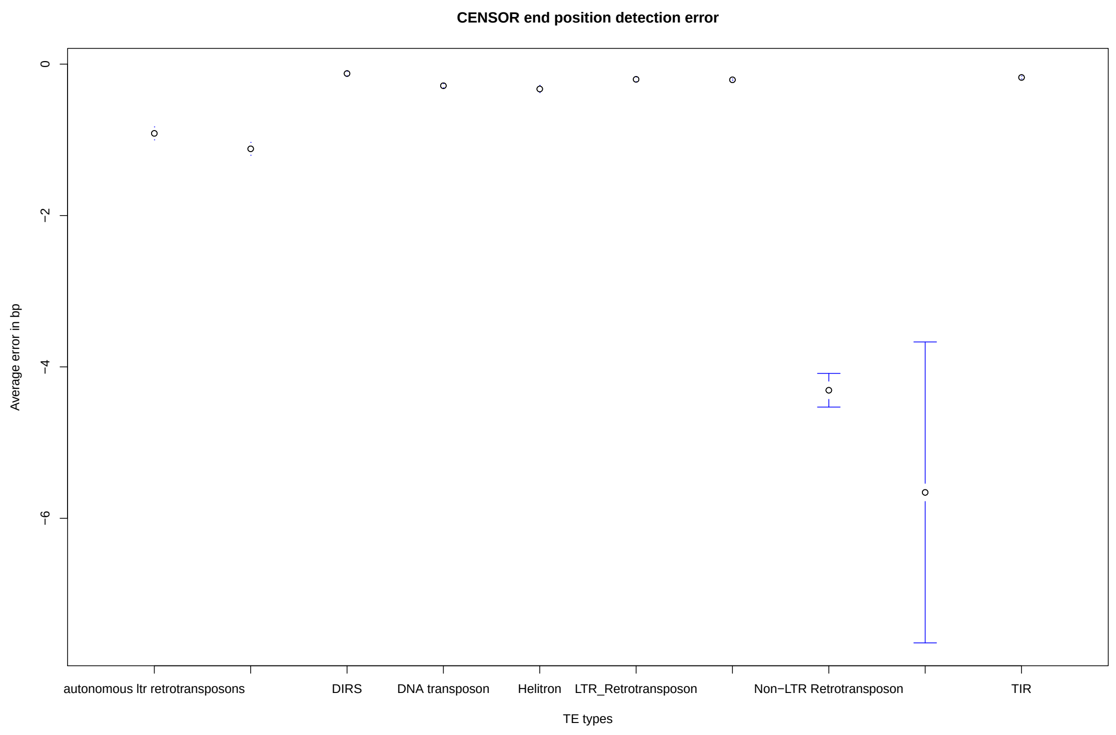

Figure 4.21: Censor "end error"

Figure 4.23 shows the average error in TE length detection according to the actual TE length. As it can be seen, the standard deviation of the error is bigger for larger values of TE length. Regardless of this, there is no statistical evidence that the TE length is related with the tool's performance in detecting the actual length, as it is shown by the correlation coefficient value of -0.12.

Figure 4.24 shows the average error in TE length detection according to the actual TE mutation percentage. Although the figure suggests an increase of error when the mutation percentage increases, there is no statistical evidence that these two variables are correlated as the correlation coefficient is -0.01.

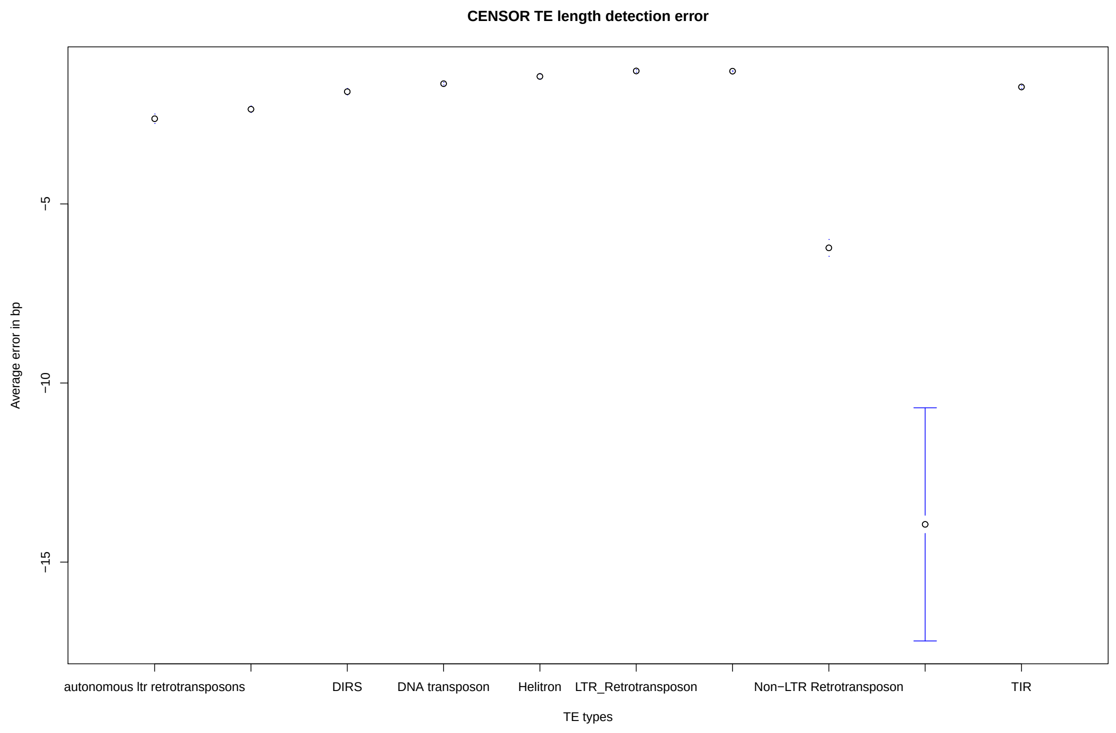

Figure 4.22: Censor TE "length error"

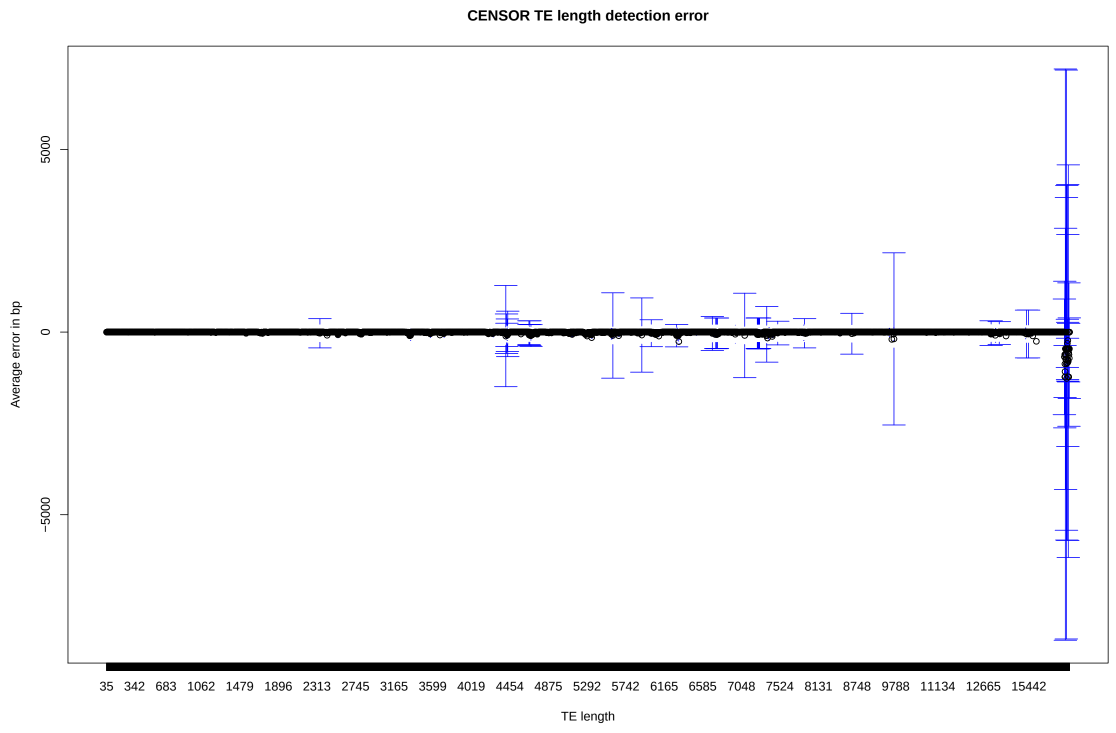

Figure 4.23: Censor’s TE length error

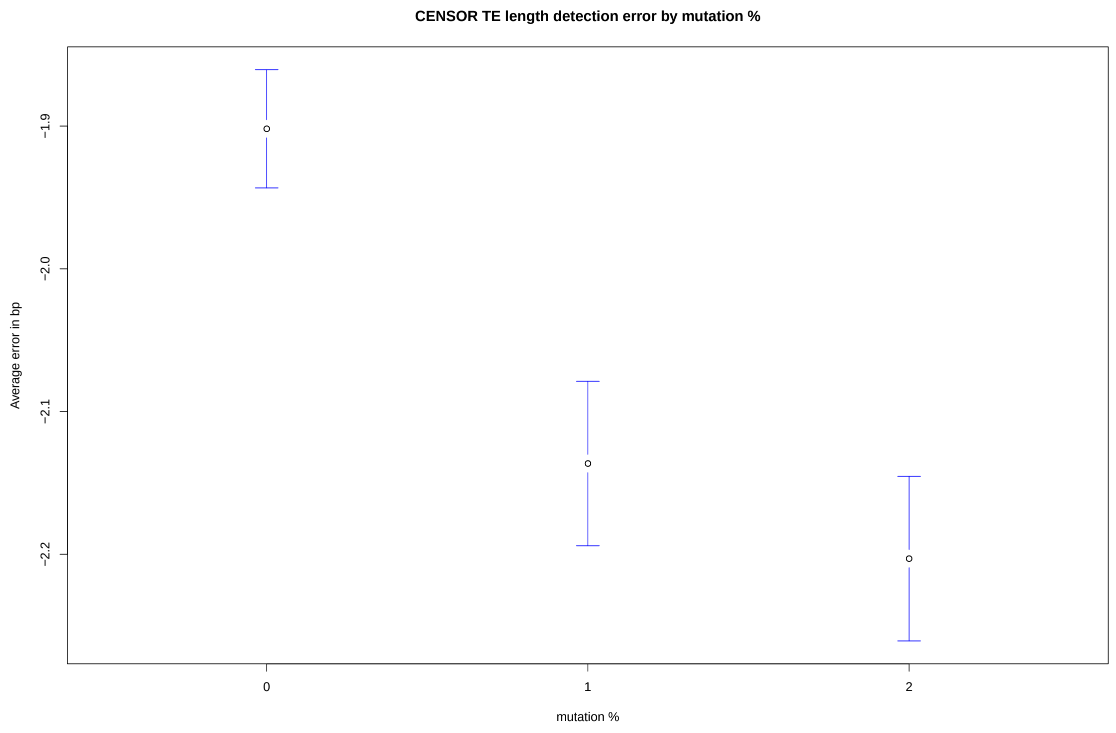

Figure 4.24: Censor's TE length error

4.2.4 LTR Finder

The results of LTR Finder assessment is shown in Figure 4.25. It shows the correctness of LTR Finder as a function of all the combination of different input parameters such as TE, genes, repetitive element and mutations percentages.

In Figure 4.26 it is shown the assessment of LTR Finder in each of the ten categories of TEs.

As it can be seen, this tool only finds DIRs, non autonomous LTR retrotransposons, non LTR retrotransposons and autonomous LTR retrotransposons. Although the actual efficiency is very poor, the best results were on the discovery of non autonomous LTR retrotransposons. Despite this fact, it performs exceptionally bad regardless of the TE type.

In Figures 4.27 and 4.28 the internal and overall correctness of this tool in finding each TE in a given sequence is shown.

Figure 4.27 shows that the local efficiency of this tool is low, with the majority of the sequence analyzed having 0 to 5% of correct TEs found per TEs found.

As it can be seen in Figure 4.28, the global efficiency of this tool is very low and almost no TEs were correctly found in the majority of the sequences. We can see that the mean and the square deviance are very low and therefore we can conclude that generically LTR Finder does not perform well.

To better understand the importance of the mutation variable in TE discovery in these sequences, we show in Figure 4.29 how LTR Finder's correctness is influenced by the mutation percentage.

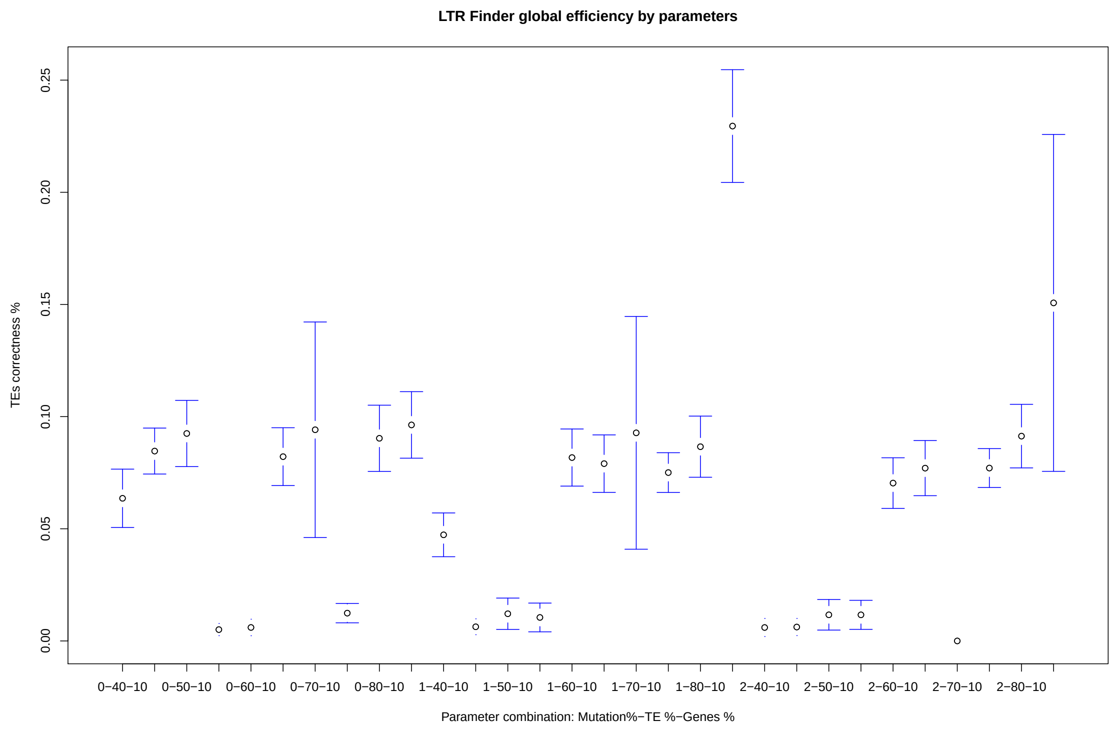

Figure 4.25: LTR Finder's accuracy plotted against the parameters combination values

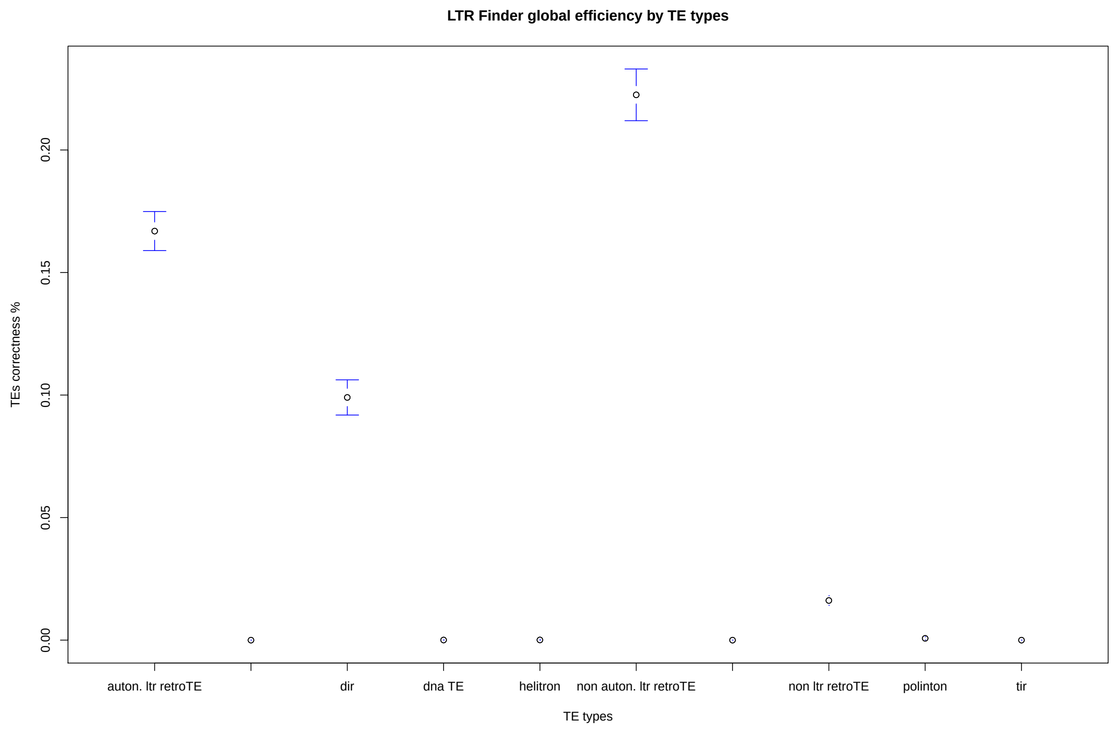

Figure 4.26: LTR Finder's accuracy according to TE types

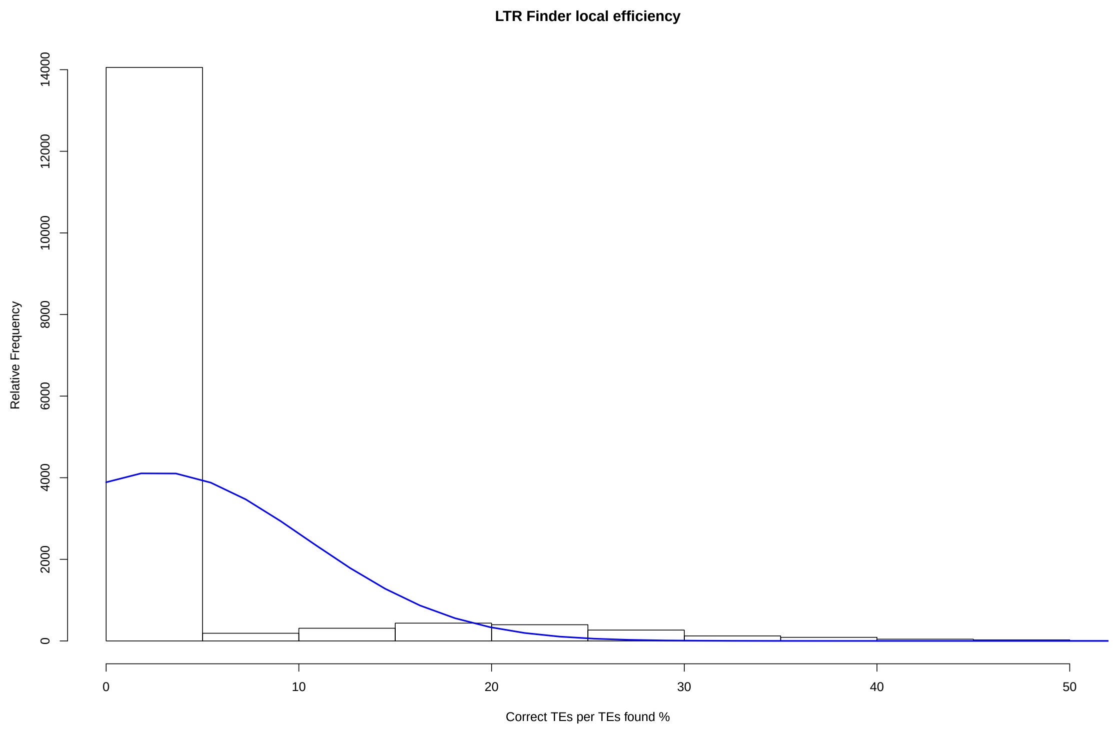

Figure 4.27: LTR Finder's internal accuracy

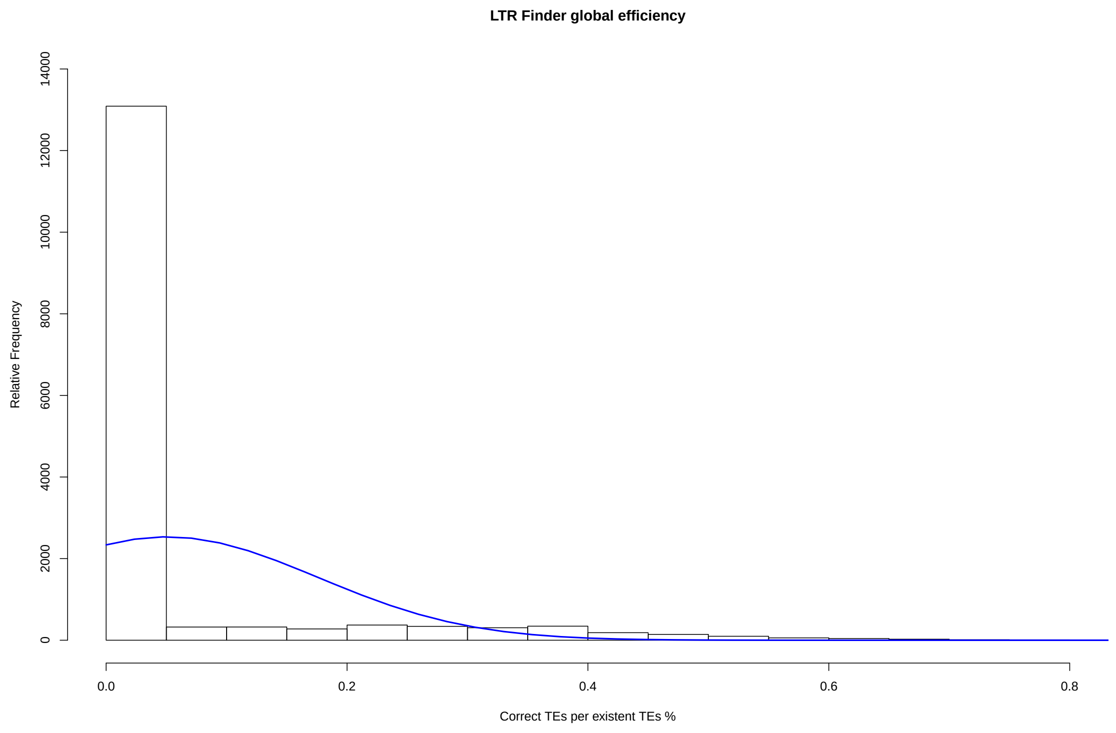

Figure 4.28: LTR Finder's actual accuracy

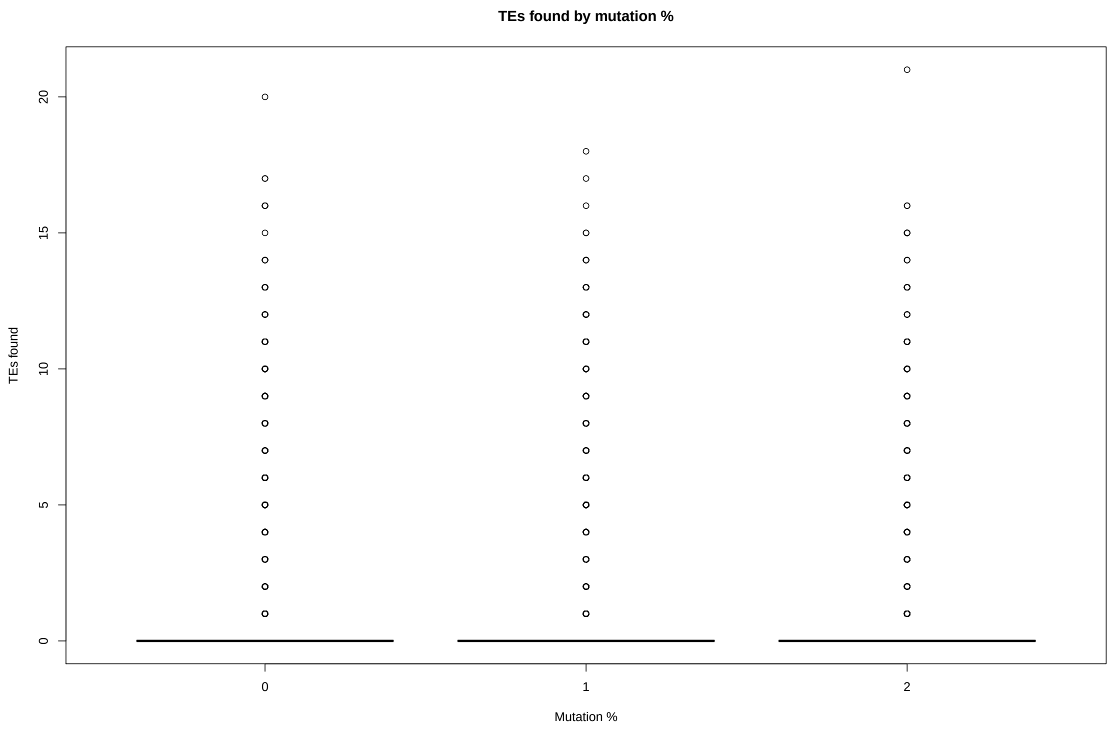

Figure 4.29: Corrrect TEs found per mutation percentage

As Figure 4.29 shows, there is minor to none differences in LTR Finder's detection capabilities when analyzing sequences with 0, 1 or 2 percent of *indel* and point mutations. The correlation coefficient between mutation percentage and the number of correct TEs found by LTR Finder is -0.02. Therefore we can assume there is no statistical evidence that mutation percentage is related to LTR Finder's accuracy.

The relevance of gene percentage in a sequence is presented in Figure 4.30.

There is no statistical evidence that gene percentage is related with the tool's performance. This is supported by the correlation coefficient value of 0.03 that demonstrate that these two variables are not linearly related.

Regarding the influence of the TE and repetitive element percentage in each sequence on the LTR Finder's performance, Figure 4.31 shows how the performance varies with the TE and repetitive element percentage.

As in the cases of genes and mutations percentage, there is no statistical evidence that the TE and repetitive elements percentage is related with the tool's performance. This is supported by the correlation coefficient value of 0.19.

Analyzing the actual detection error is yet another problem that TE detection tools face. Regarding this, we show in Figures 4.32, 4.33 and 4.34 how well the LTR Finder tool performs.

As Figure 4.32 shows, the "begin position" error of LTR Finder tool ranges between 0bp and 50bp in every TE category with the exception of Helitron TEs. In this case the error is larger than 300bp in average. In terms of "end error" shown on Figure 4.33, it can be seen that the average

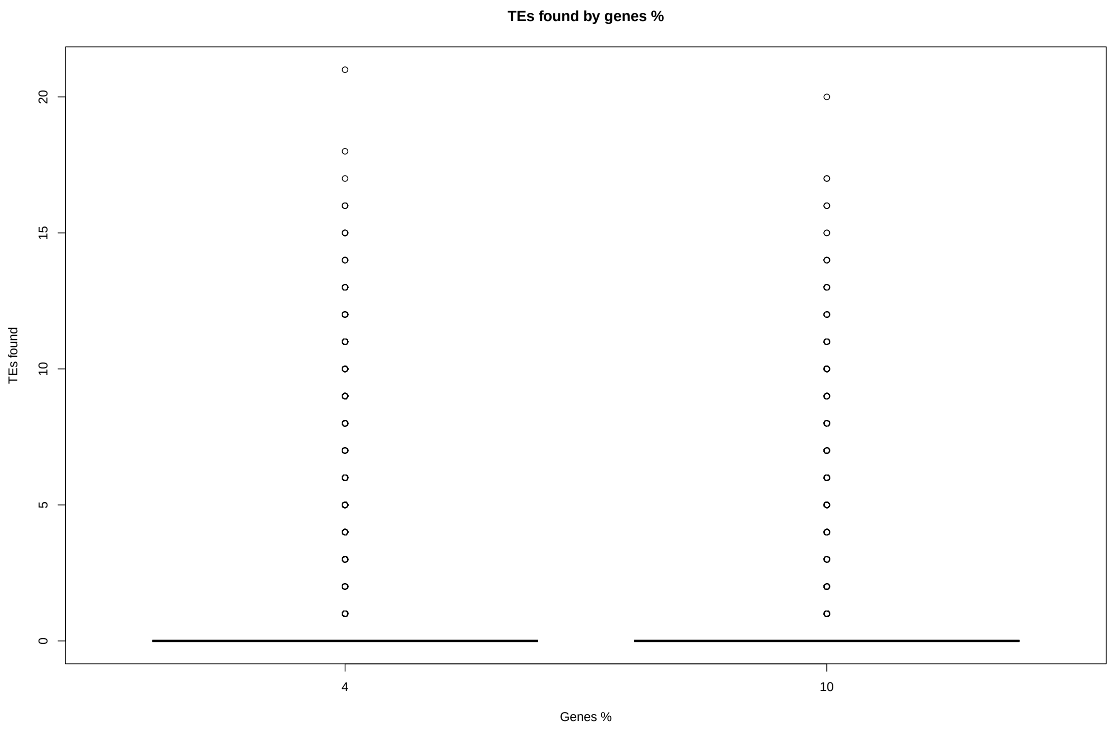

Figure 4.30: Corrrect TEs found according to gene percentage

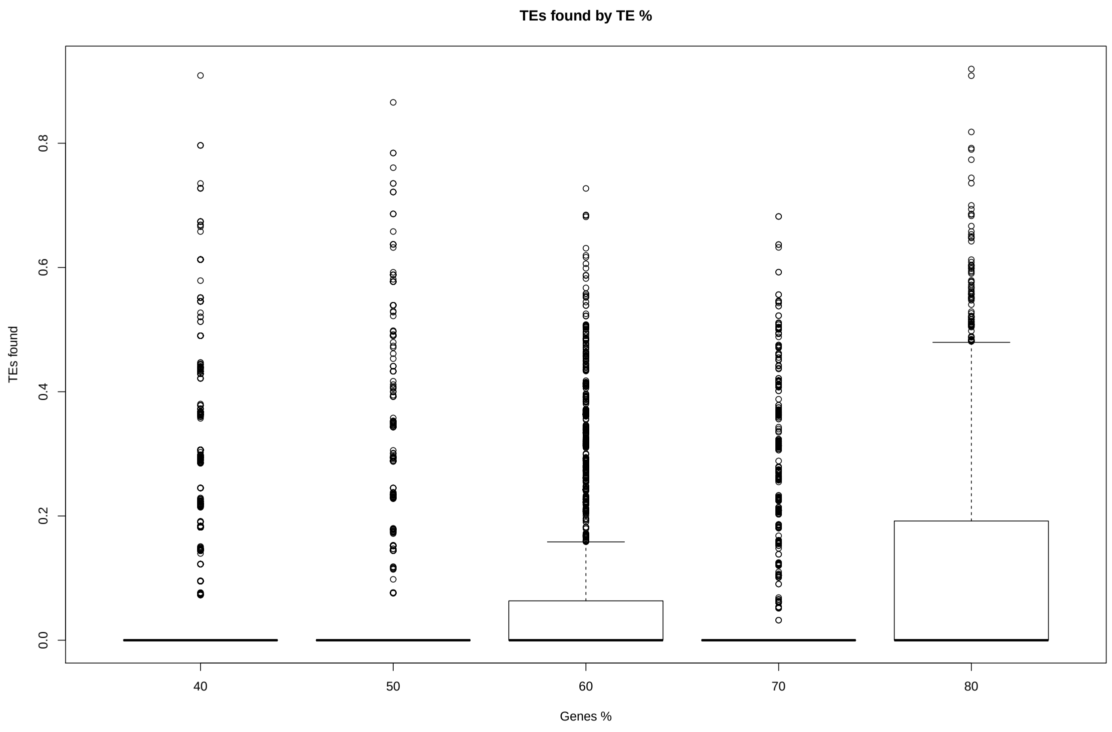

Figure 4.31: Corrrect TEs found according to TE and repetitive elements percentage

error varies from 0bp to 200bp. Once again, this tool is not particularly accurate regarding Helitron detection. Figure 4.34 shows that the error in length varies from 0bp to 200bp in all TE categories except Helitron TEs.

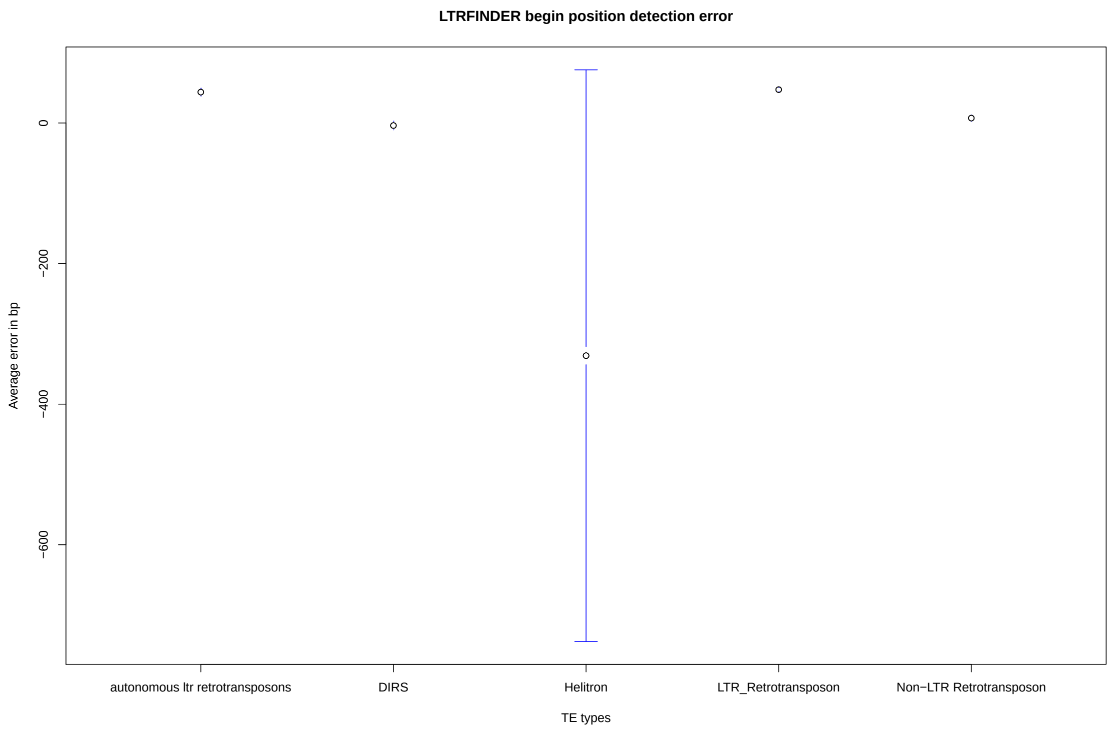

Figure 4.32: LTR Finder "begin error"

Figure 4.35 shows the average error in TE length detection according to the actual TE length. As it can be seen from the figure, that there is no relation whatsoever between the TE length and the TE length error. This fact is corroborated by the correlation coefficient of 0.1.

Figure 4.36 shows the average error in TE length detection according to the actual TE length. There is no relation between the TE length error and the mutation percentage present in the sequences, as is suggested by the correlation coefficient value of 0.01.

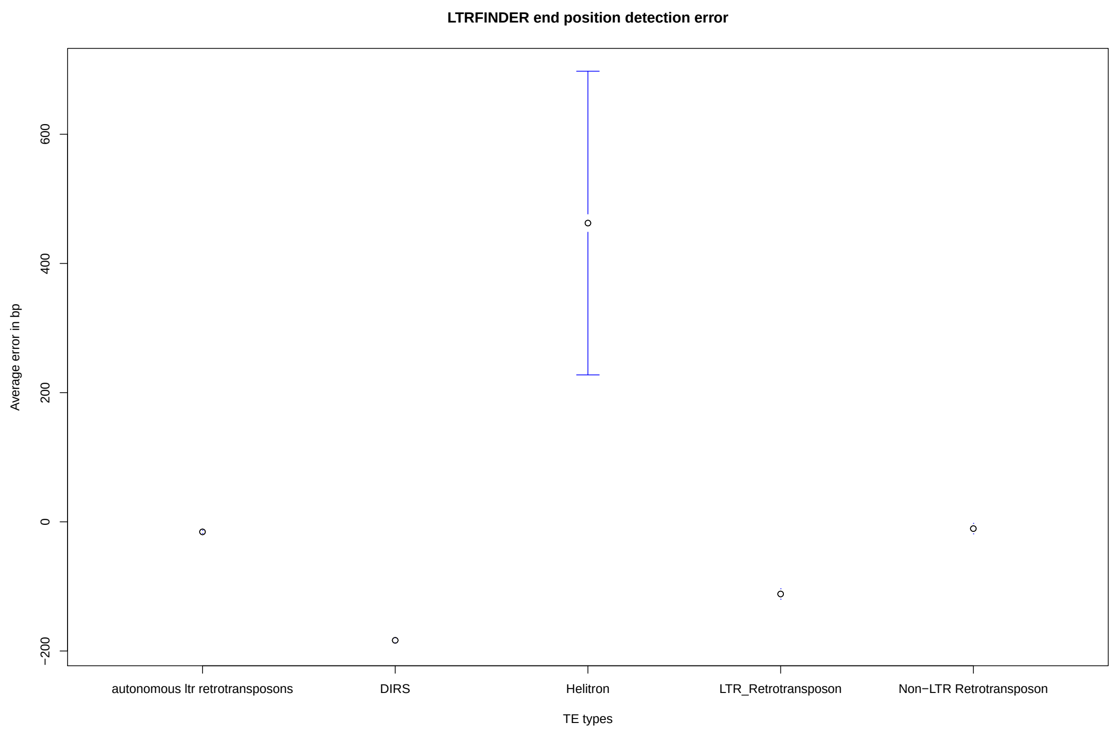

Figure 4.33: LTR Finder "end error"

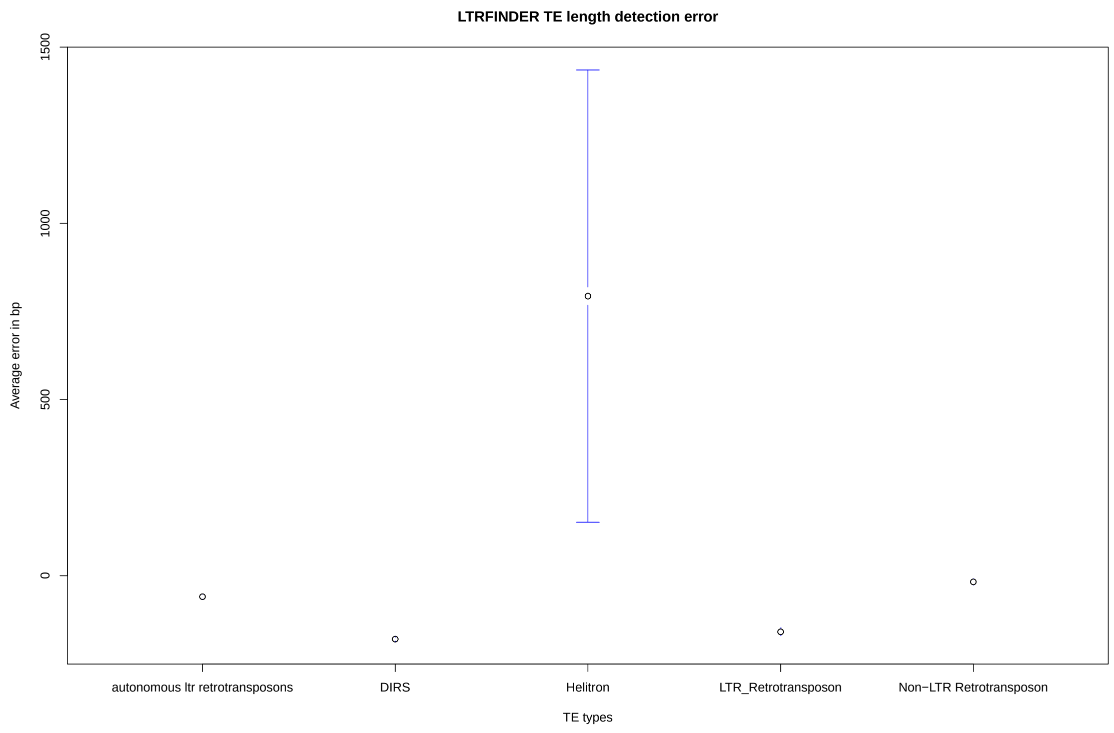

Figure 4.34: LTR Finder TE "length error"

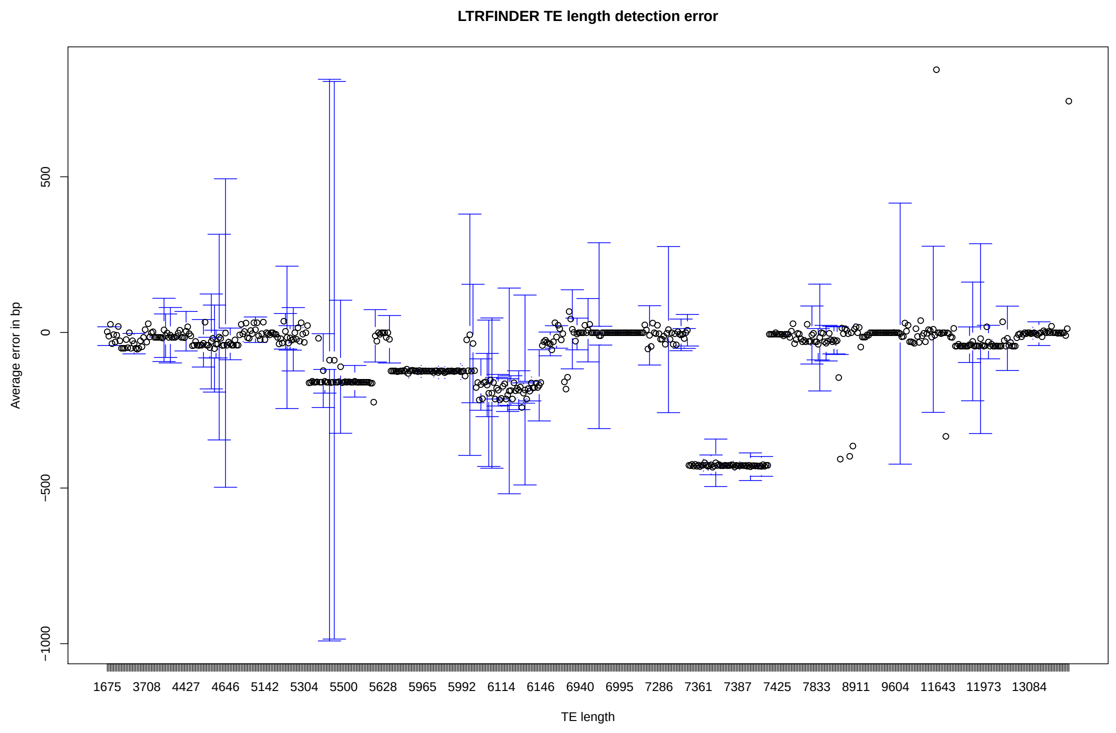

Figure 4.35: LTR Finder’s TE length error

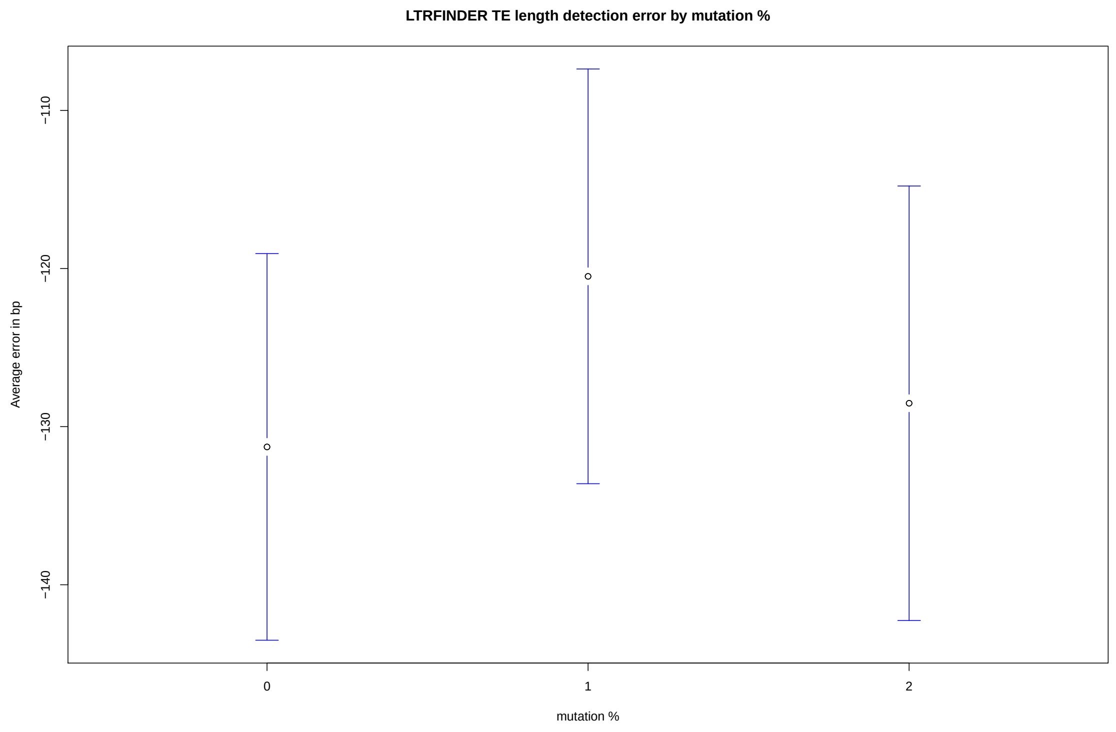

Figure 4.36: LTR Finder’s TE length error

4.2.5 PILER

The results of PILER assessment is shown in Figure 4.37. It shows the correctness of PILER as a function of all the combination of different input parameters such as TE, genes, repetitive element and mutations percentages. This representation suggests a strong evidence that the mutation percentage has a negative effect regarding the TE detection capabilities of this tool.

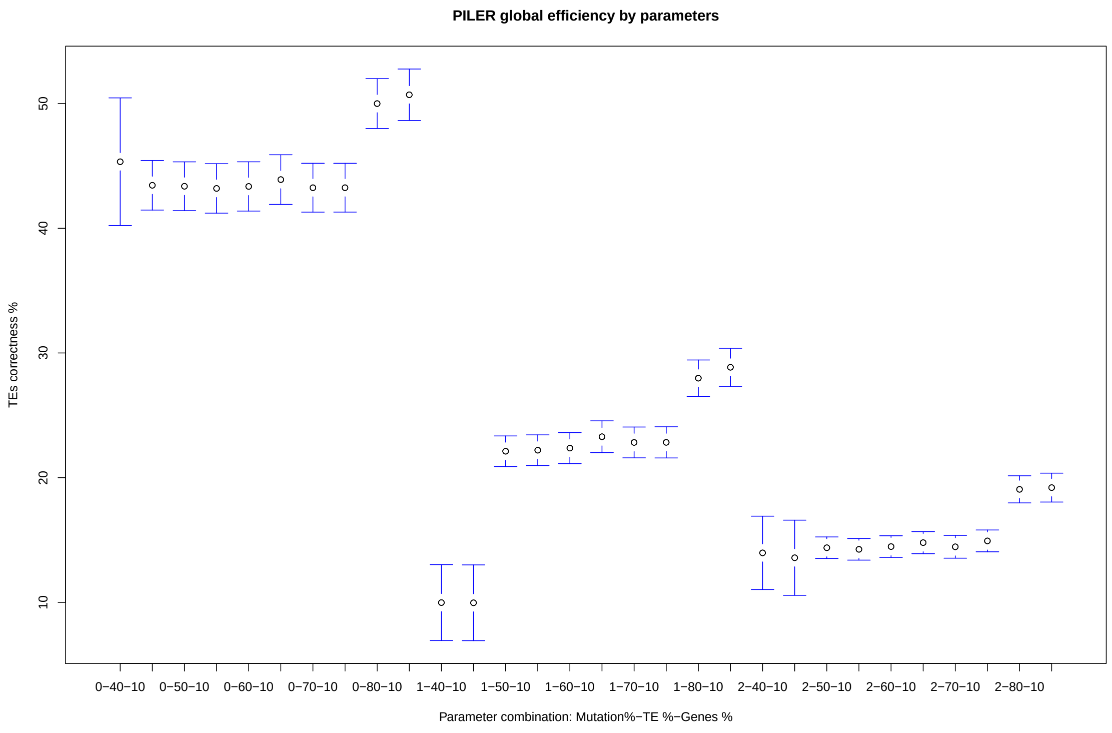

Figure 4.37: PILERS's accuracy plotted against the parameters combination values

In Figure 4.38 it is shown the assessment of PILER in each of the ten categories of TEs.

As it can be seen, this tool only lacks the ability of identifying autonomous LTR retrotransposons and has poor results regarding non autonomous non-LTR retrotransposon detection. As for the rest of the categories, the results vary from roughly 25% on DNA TEs, Polintons and TIRs up to almost 50% on autonomous non-LTR retrotransposons.

In Figures 4.39 and 4.40 the internal and overall correctness of this tool in finding each TE in a given sequence is shown.

Figure 4.39 shows that the local efficiency of this tool ranges from 0% to roughly 70% of correct TEs found per TEs found, being the majority of the accessments under 40%.

Figure 4.40 shows the global efficiency of this tool varying from very low (almost 0%) to very high (above than 80%). Despite this, the majority of the sequences analyzed had an efficiency of 40% or less.

To better understand the importance of the mutation variable in TE discovery in these sequences, we show, in Figure 4.41, how PILER's correctness is influenced or not by the mutation percentage.

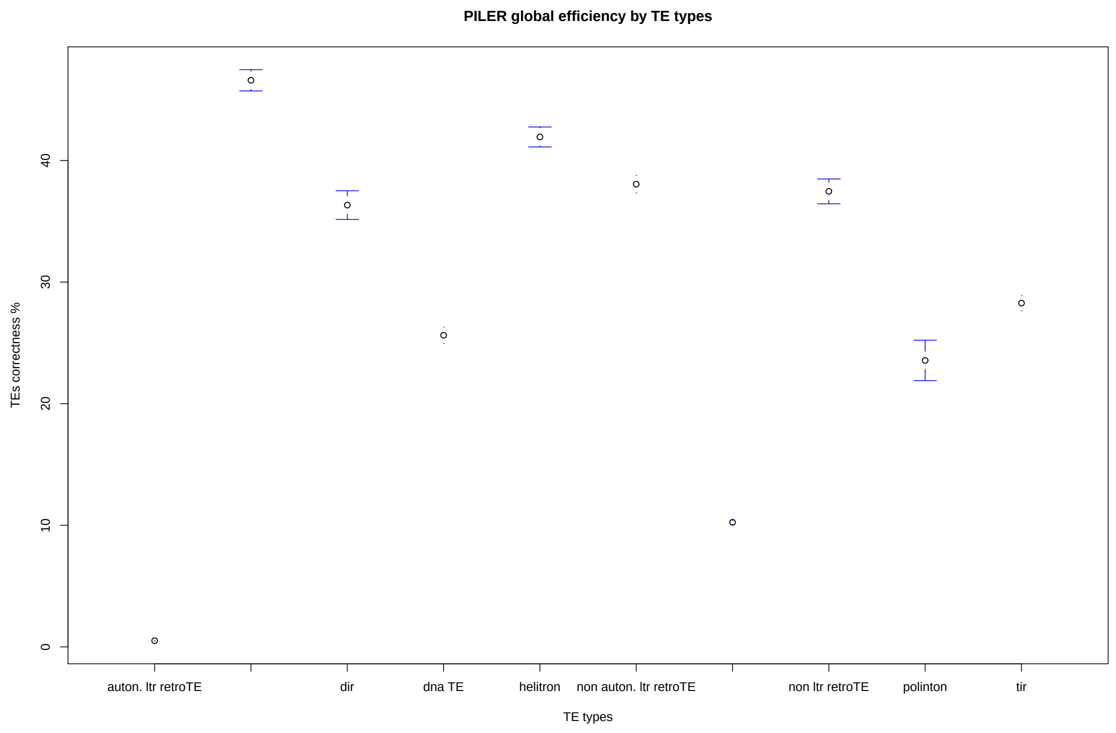

Figure 4.38: PILER's accuracy according to TE types

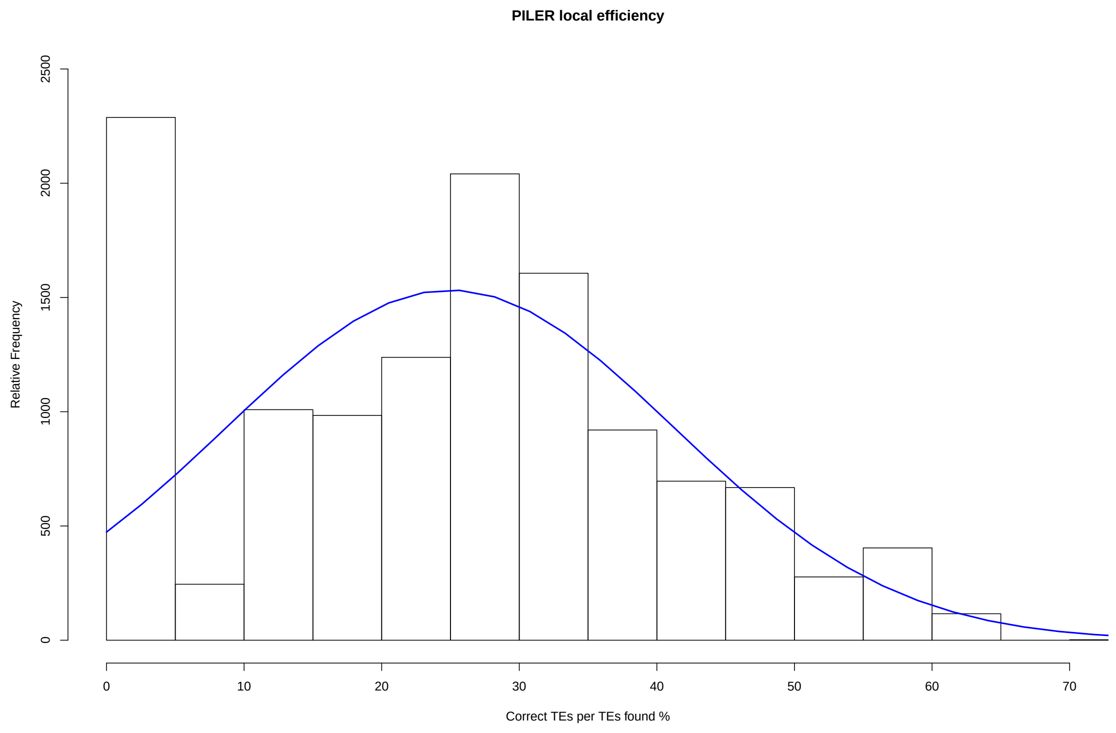

Figure 4.39: PILER's internal accuracy

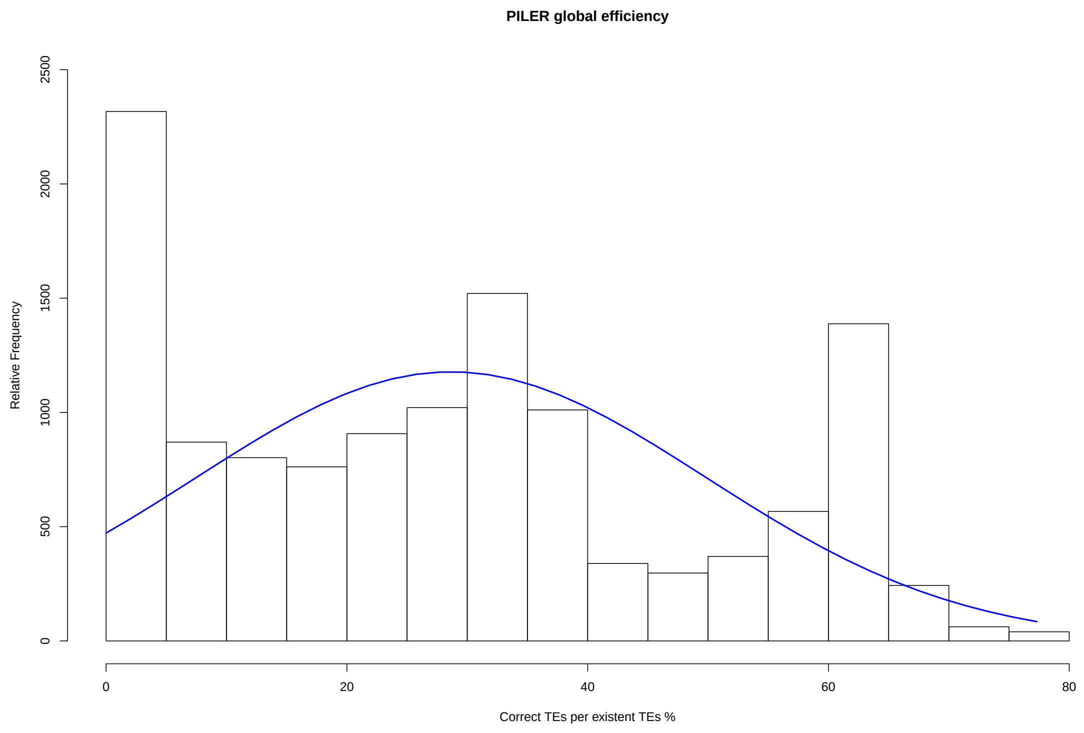

Figure 4.40: PILER's actual accuracy

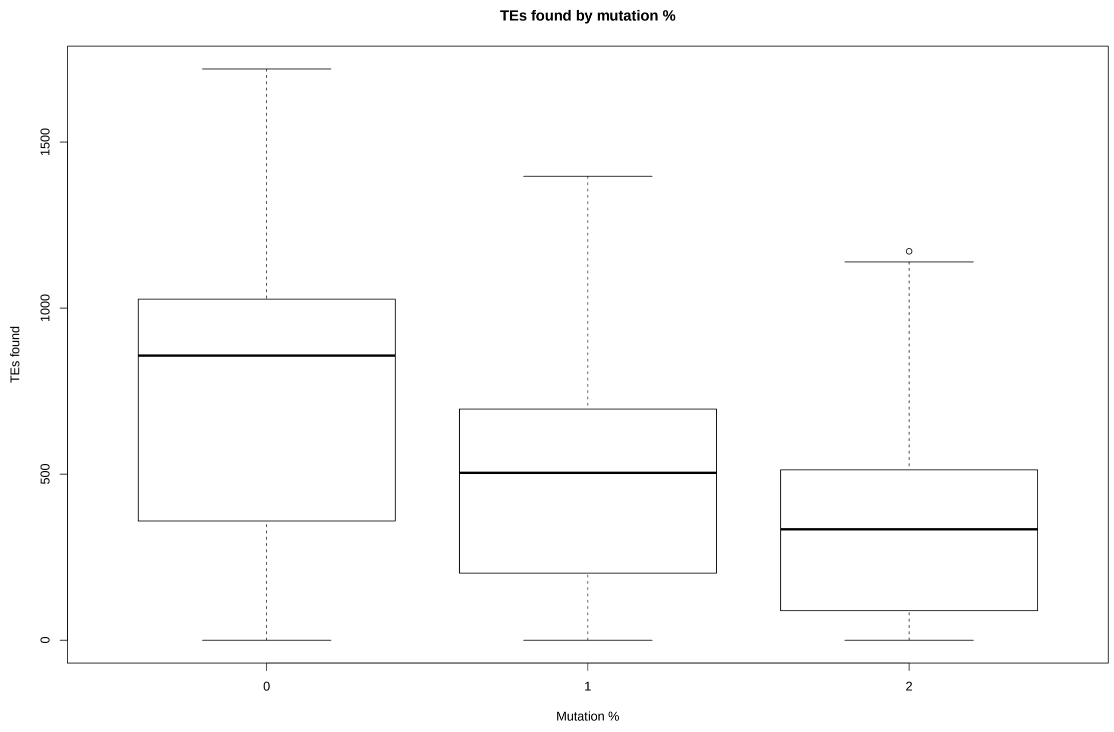

Figure 4.41: Corrrect TEs found per mutation percentage

As Figure 4.41 shows, there is an effective decrease of efficiency of this tool when the mutation percentage increases. This is supported by the correlation coefficient between mutation percentage and the number of correct TEs found by PILER: -0.57. Therefore we can assume that statisticaly this two variables are correlated and therefore PILER's results are influenced directly by the mutation percentage.

The relevance of gene percentage in a sequence is presented in Figure 4.42.

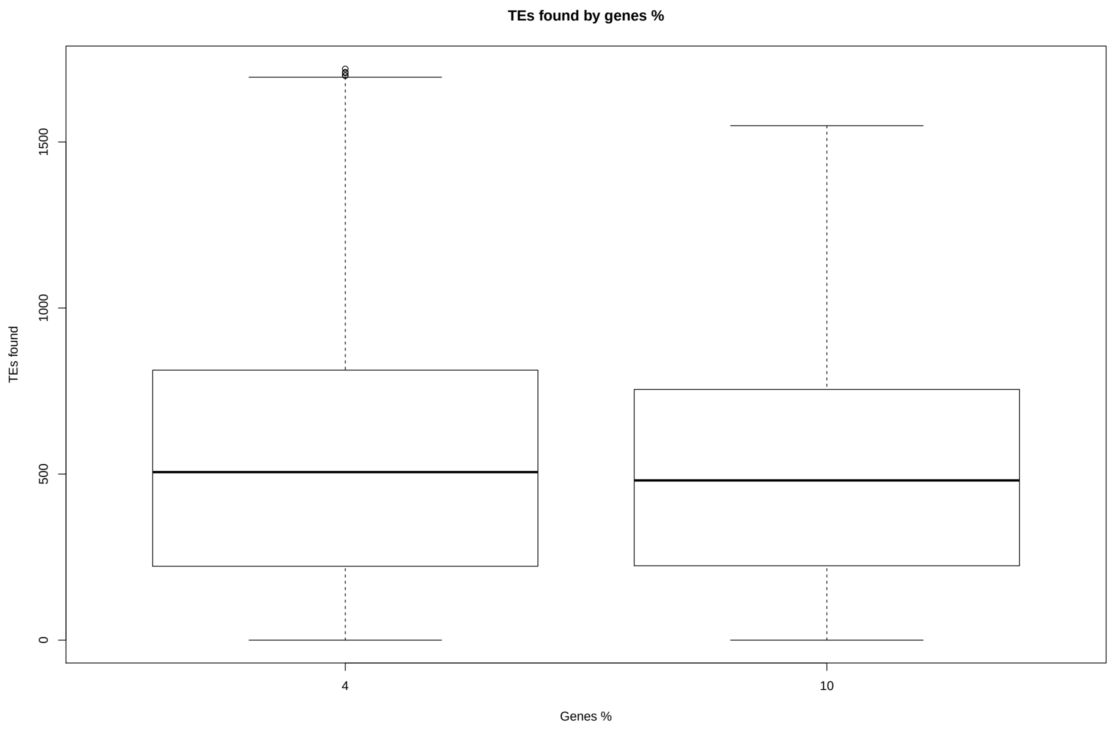

Figure 4.42: Corrrect TEs found according to gene percentage

There is no statistical evidence that gene percentage is related with the tool's performance. This is supported by the correlation coefficient value of -0.03 that demonstrate that these two variables are not linearly related.

Regarding the influence of the TE and repetitive elements percentage in each sequence on the LTR Finder's performance, in Figure 4.43 it is show how it varies with the TE percentage.

In the same situation of genes and mutations percentage, there is no statistical evidence that the TE and repetitive elements percentage is related with the tool's performance. This is supported by the correlation coefficient value of 0.05.

Analyzing the actual detection error is yet another problem that TE detection tools face. Regarding this, we show in Figures 4.44, 4.45 and 4.46 how well the PILER tool performs.

As Figure 4.44 shows, the "begin position" error of PILER tool ranges between 0bp and 2bp in every TE category with the exception of DIRS. In this case the error is above 4bp in average. In terms of "end error" show on 4.45, it is suggested that the average error varies from 2bp to 6bp. One more time, this tools is not particularly accurate regarding DIRS detection, as the average

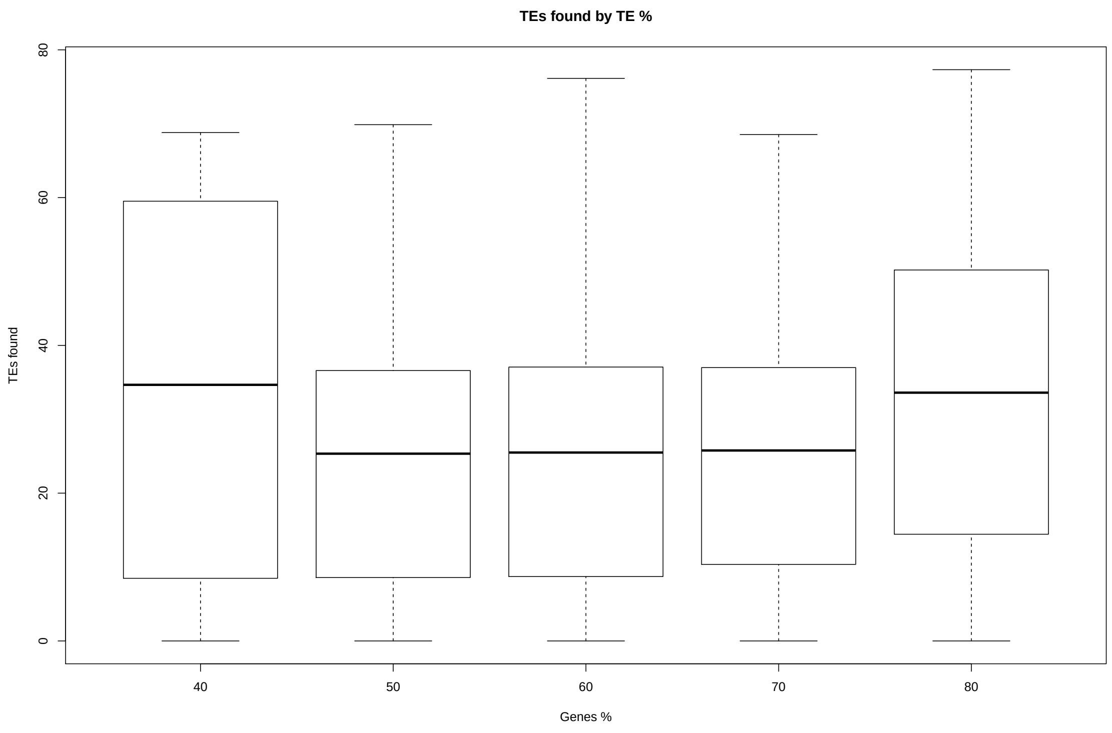

Figure 4.43: Corrrect TEs found according to TE and repetitive elements percentage

error is almost 9bp. Figure 4.46 suggests that the error in length varies from 2bp to 8bp in all TE categories except DIRS.

Figure 4.47 shows the average error in TE length detection according to the actual TE length. There is no correlation between TE length and the TE length error as the correlation of value 0.03 shows. Regardless the figure shows that the standard deviation of the error increases as the TE length increases.

Figure 4.48 shows the average error in TE length detection according to the mutation percentage of the sequence. There is no correlation between these two variables as the correlation coefficient demonstrates: -0.1.

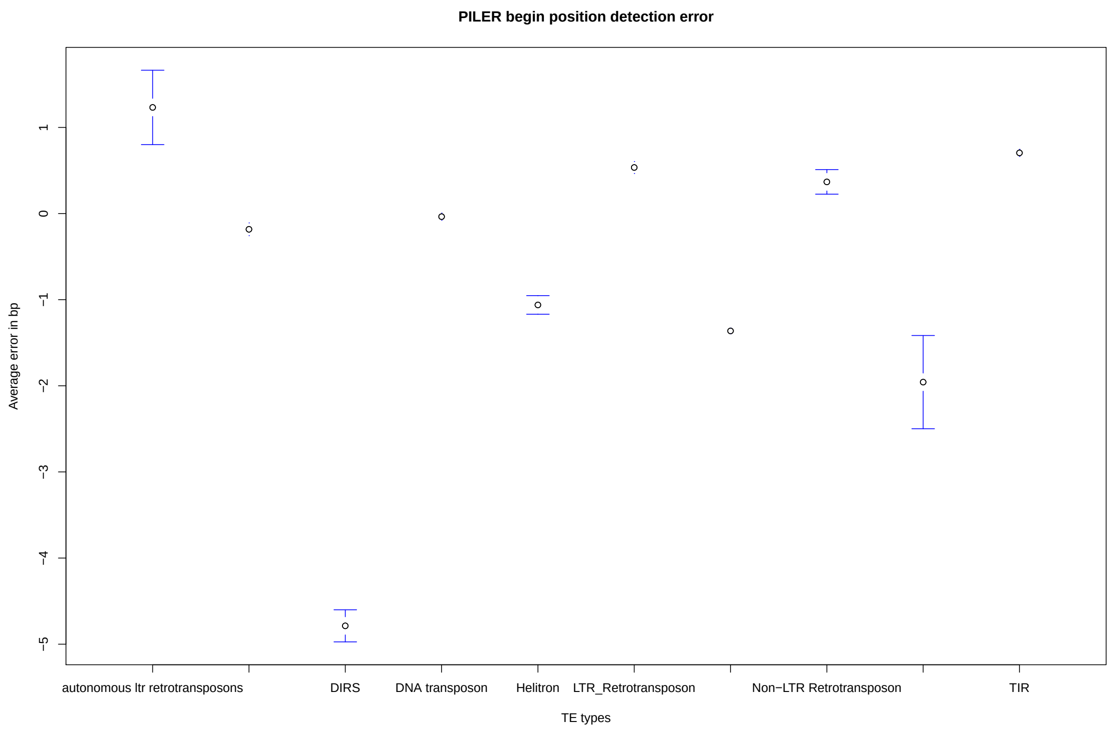

Figure 4.44: PILER "begin error"

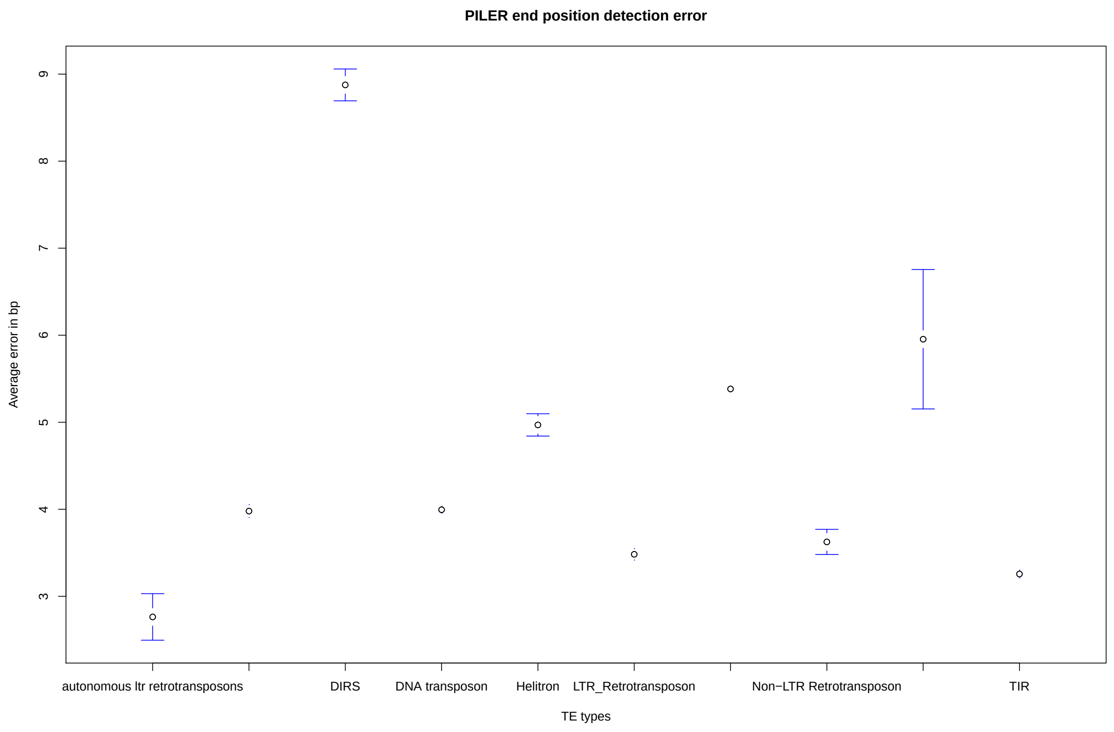

Figure 4.45: PILER "end error"

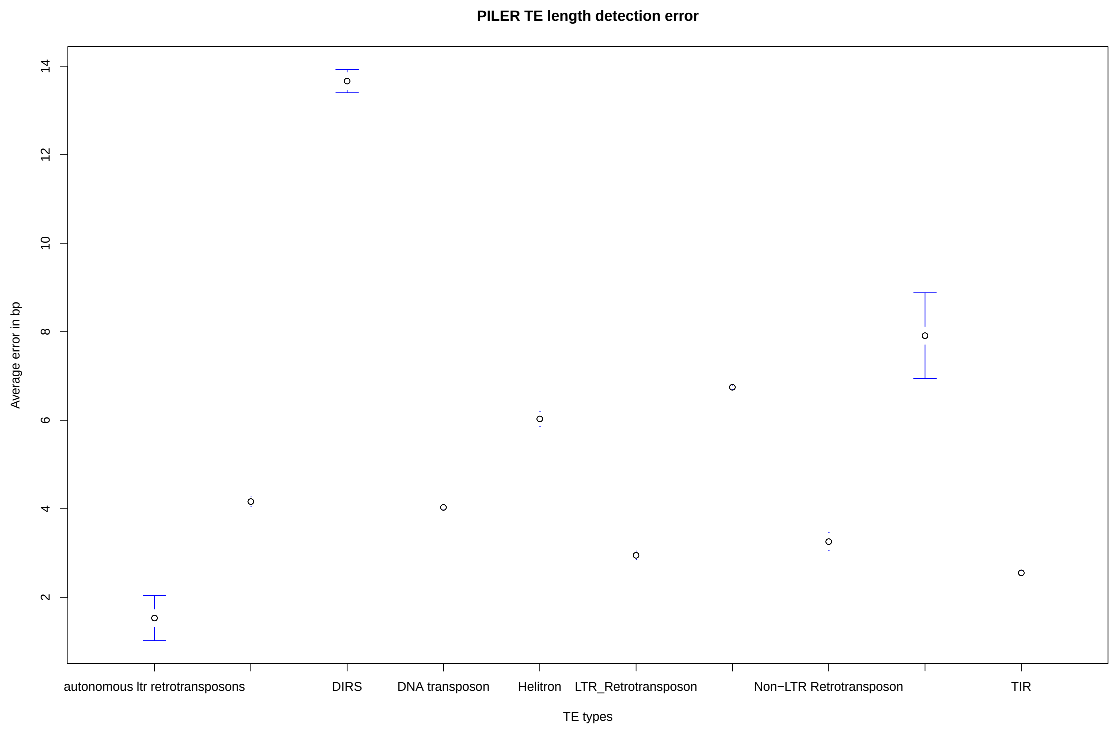

Figure 4.46: PILER TE "length error"

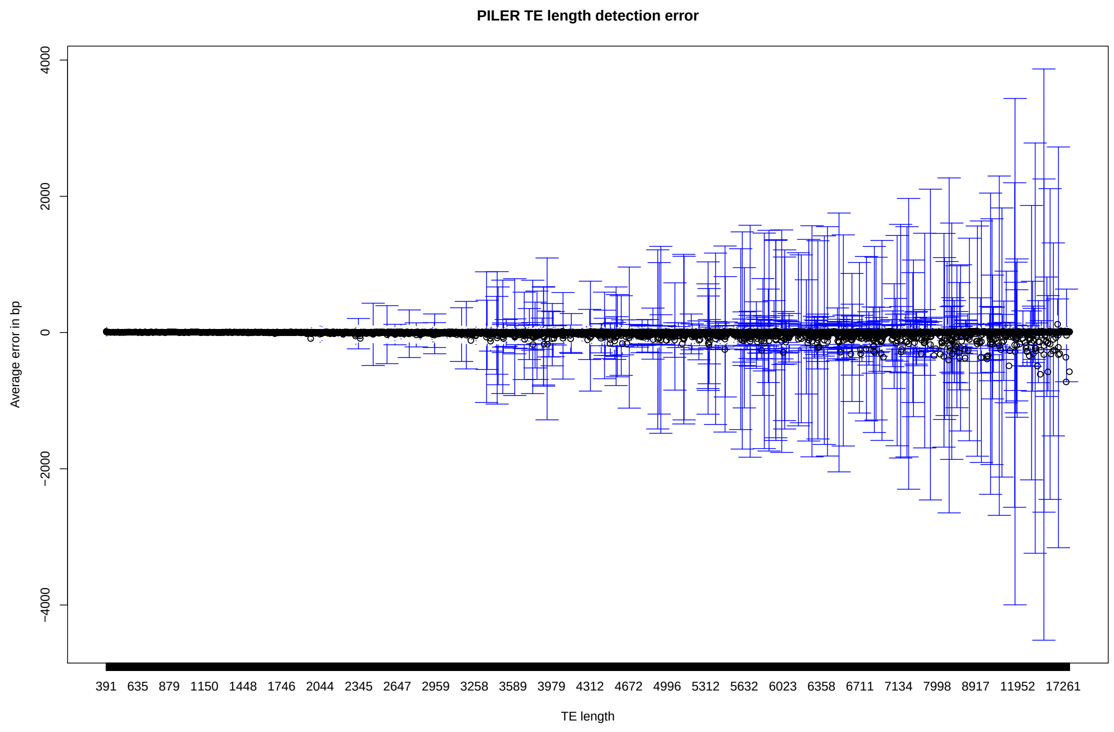

Figure 4.47: PILER's TE length error

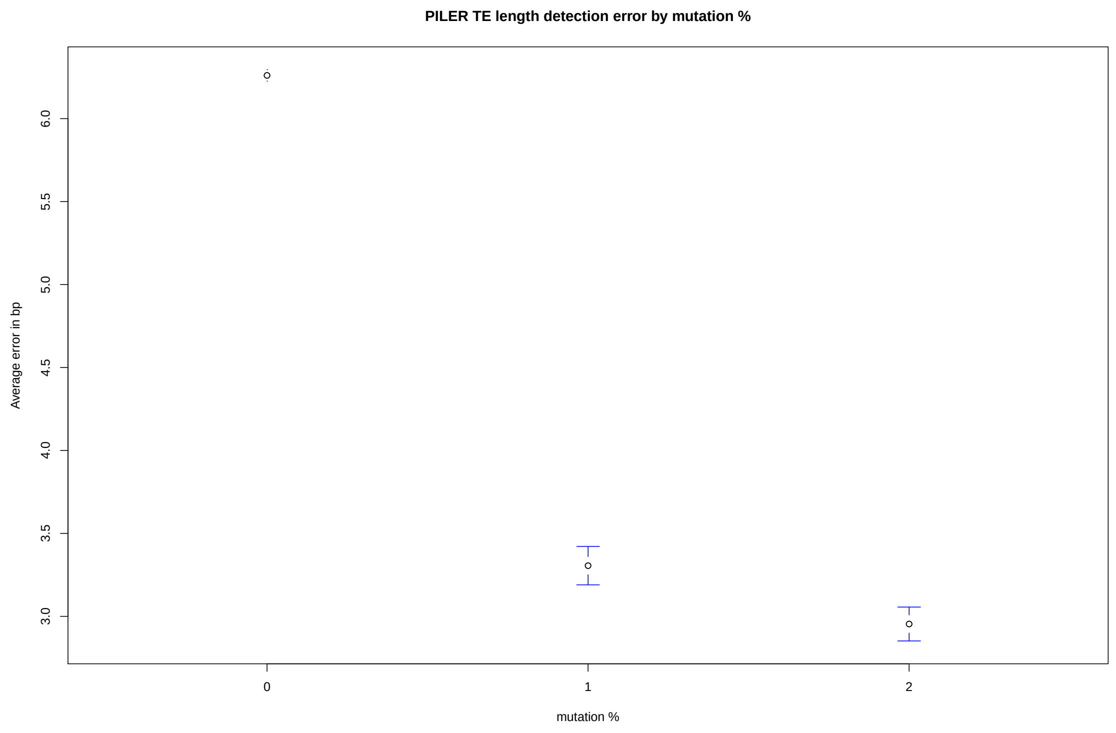

Figure 4.48: PILER's TE length error

4.2.6 RepeatMasker

The results of RepeatMasker assessment is shown in Figure 4.49. It shows the correctness of RepeatMasker as a function of all the combination of different input parameters such as TE, genes, repetitive element and mutations percentages.

In Figure 4.50 it is shown the assessment of the tool in each of the ten categories of TEs.

As it becomes evident, the majority of the TE types have an accuracy which ranges between 50% to 60%. The exceptions are DIRS, Helitrons and non-LTR retrotransposons which range from 30% to roughly 40%. This tool lacks the ability to identify Polinton TEs has it's accuracy in this particular TE type is less than 5%.

In Figures 4.51 and 4.52 the internal and overall correctness of this tool in finding each TE in a given sequence is shown.

Figure 4.51 shows that the local efficiency varies from 0% to roughly 49% and in the majority of the analyzed sequences there is a result of 25% or less accuracy which means that this tool discovers a huge number of false positive TEs.

As it can be seen in Figure 4.52, the global efficiency of this tool ranges from 0% to roughly 75%, being the most sequences assessments score above 40%.

To better understand the importance of the mutation variable in TE discovery in these sequences, we show in Figure 4.53 how RepeatMasker's correctness is influenced or not by the mutation percentage.

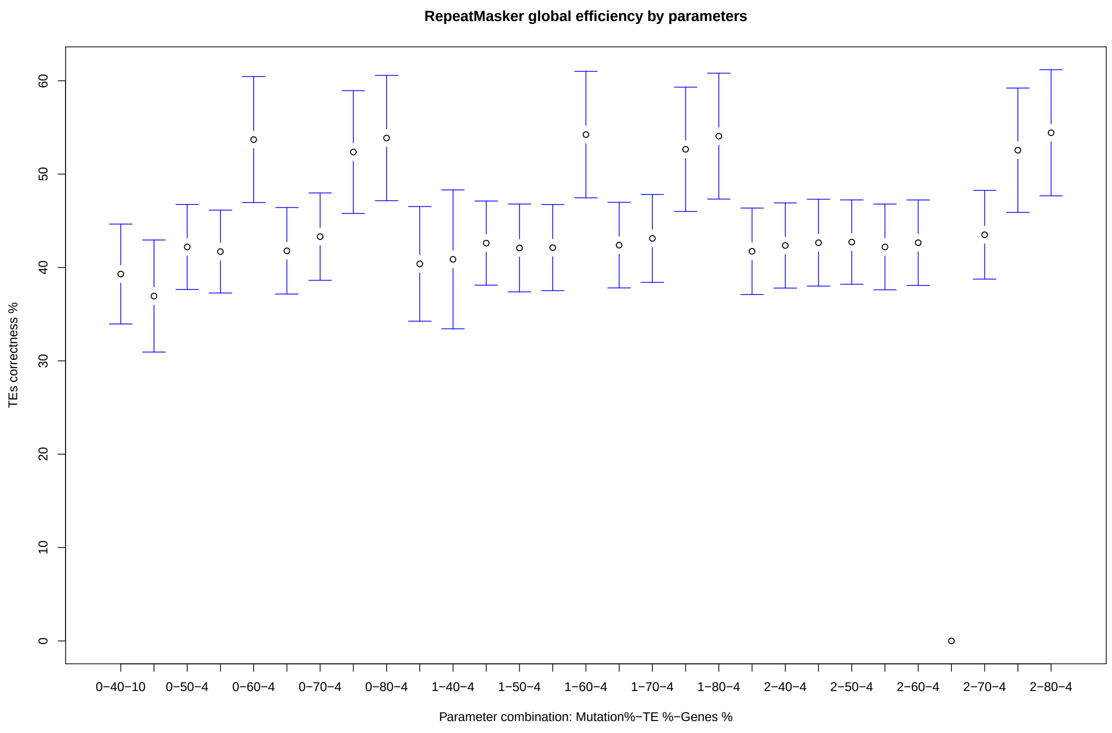

Figure 4.49: RepeatMasker's accuracy plotted against the parameters combination values

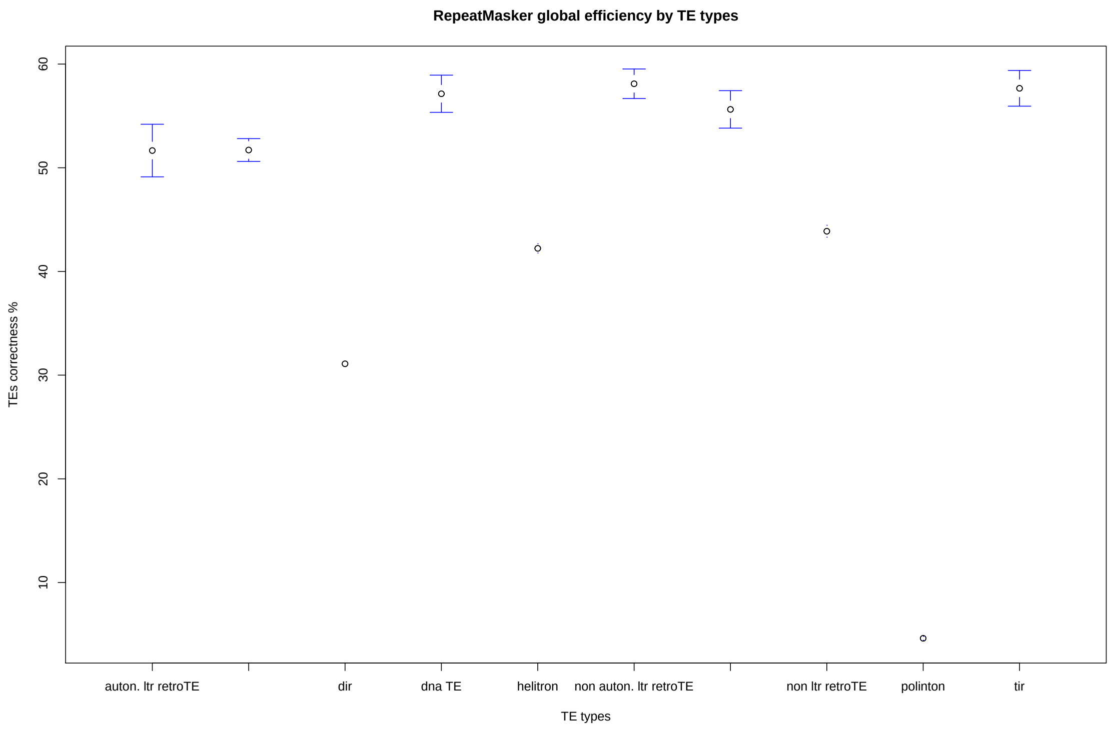

Figure 4.50: RepeatMasker's accuracy according to TE types

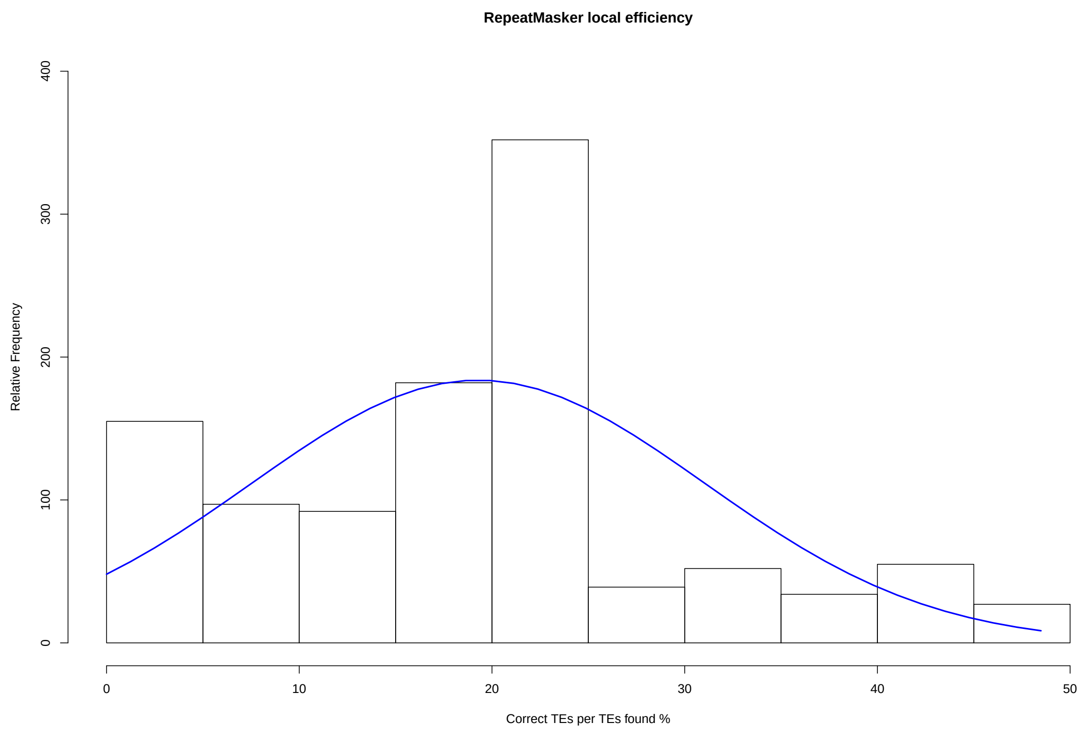

Figure 4.51: RepeatMasker's accuracy

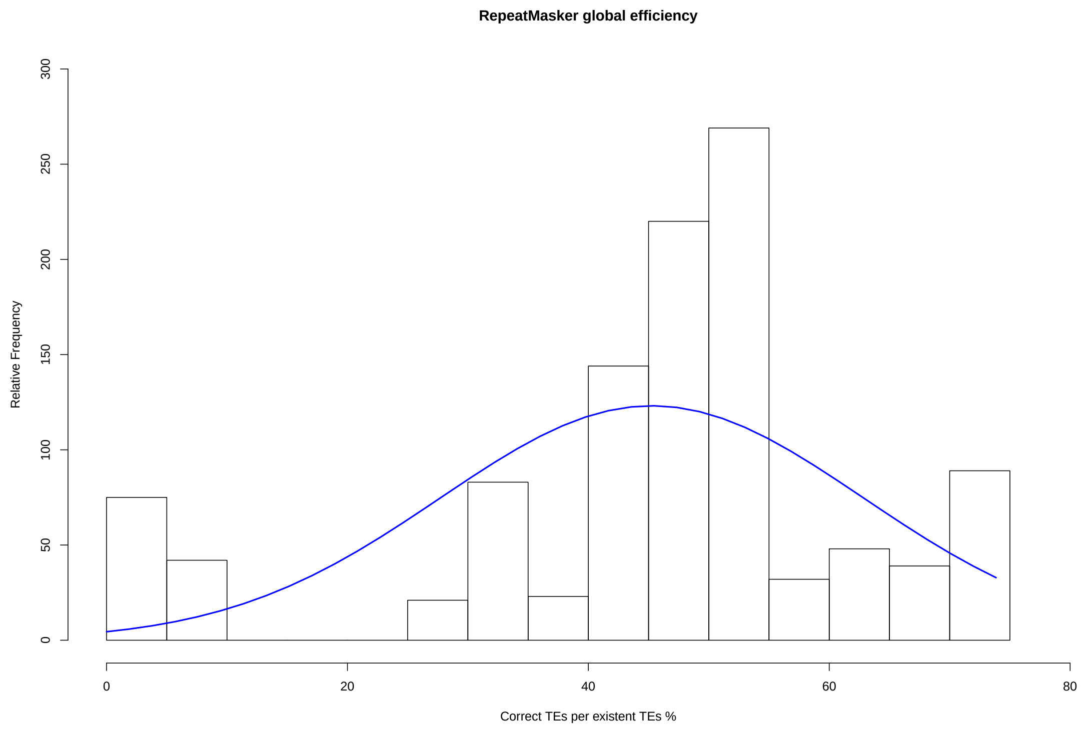

Figure 4.52: RepeatMasker's actual accuracy

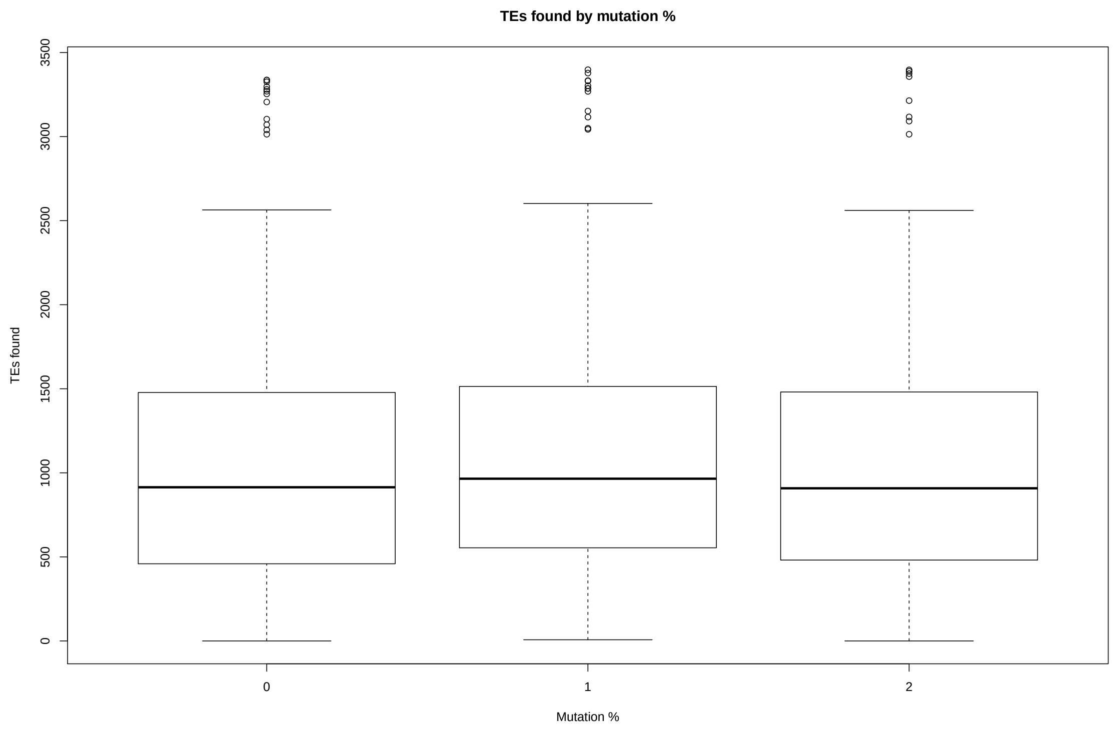

Figure 4.53: Corrrect TEs found per mutation percentage

As Figure 4.53 shows, there is minor to none differences in RepeatMasker's detection capabilities when analyzing sequences with 0, 1 or 2 percent of *indel* and point mutations. The correlation coefficient between mutation percentage and the number of correct TEs found by RepeatMasker is -0.01. Therefore we can assume there is no statistical evidence that mutation percentage is correlated to RepeatMasker's accuracy.

The relevance of gene percentage in a sequence is presented in Figure 4.54.

There is no statistical evidence that gene percentage is related with the tool's performance. This is supported by the correlation coefficient value of -0.09 that demonstrate that these two variables are not linearly related.

Regarding the influence of the TE and repetitive elements percentage in each sequence on the RepeatMasker's performance, in Figure 4.55 it is show how it varies with the TE percentage.

In the same situation of genes and mutations percentage, there is no statistical evidence that the TE and repetitive elements percentage is related with the tool's performance. This is supported by the correlation coefficient value of 0.21.

Analyzing the actual detection error is yet another problem that TE detection tools face. Regarding this, we show in Figures 4.56, 4.57 and 4.58 how well the RepeatMasker tool performs.

As Figure 4.56 shows, the "begin position" error of RepeatMasker tool ranges between 0bp and 2bp in every TE category with the exception of Helitrons and Polintons. In these cases the error is above 6bp and 8bp respectively. In terms of "end error" show on 4.57, it shows that the average error varies from 0bp to 2bp. One more time, this tools is not particularly accurate regarding

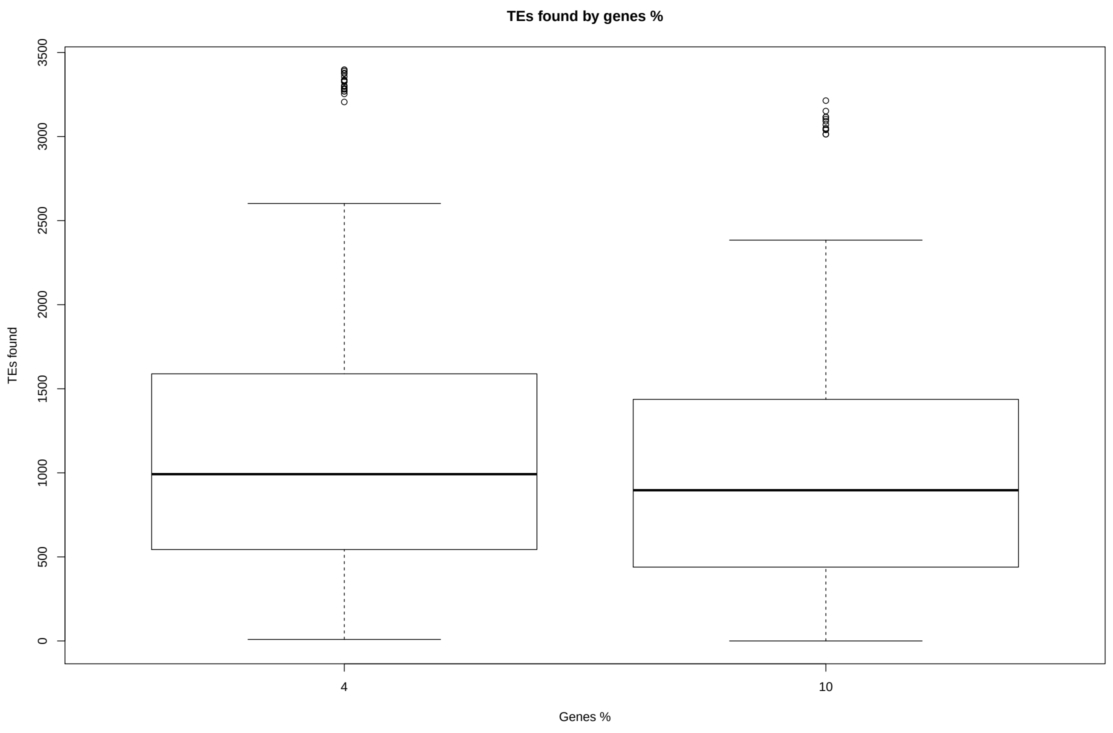

Figure 4.54: Corrrect TEs found according to gene percentage

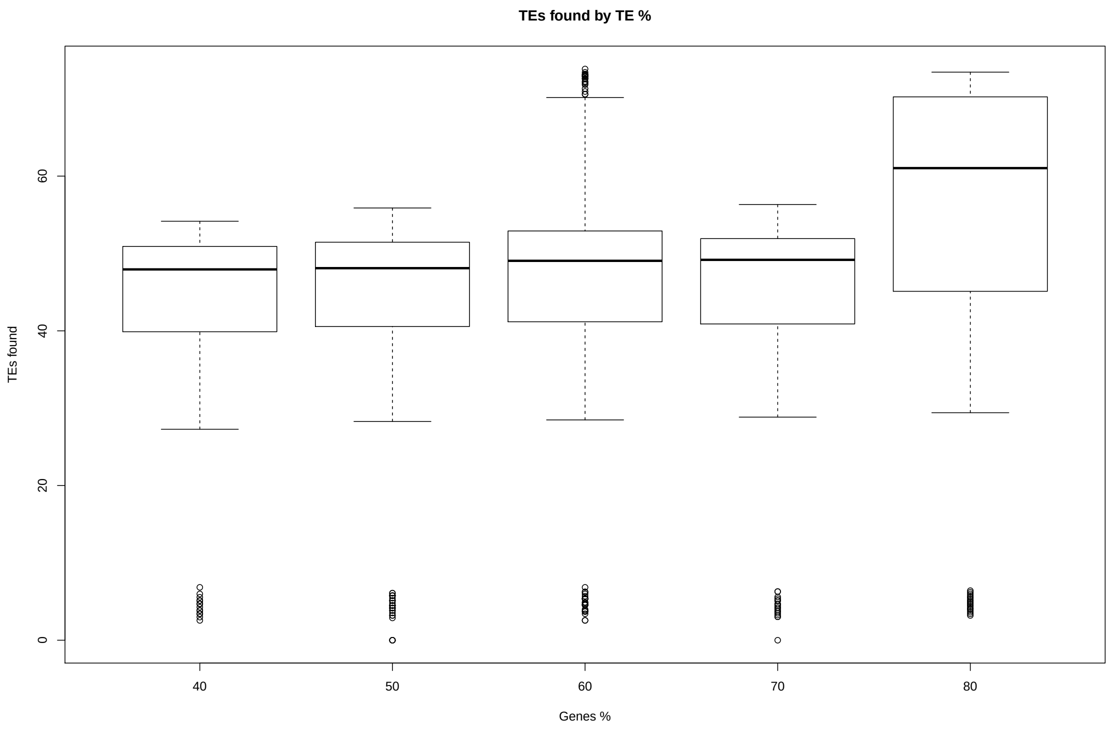

Figure 4.55: Corrrect TEs found according to TE and repetitive elements percentage

Polintons detection, as the average error is almost 9bp. Figure 4.58 shows that the error in length varies from 0bp to 5bp in all TE categories except Helitrons and Polintons, with the second having the worst result of all, with an average error of more than 16bp.

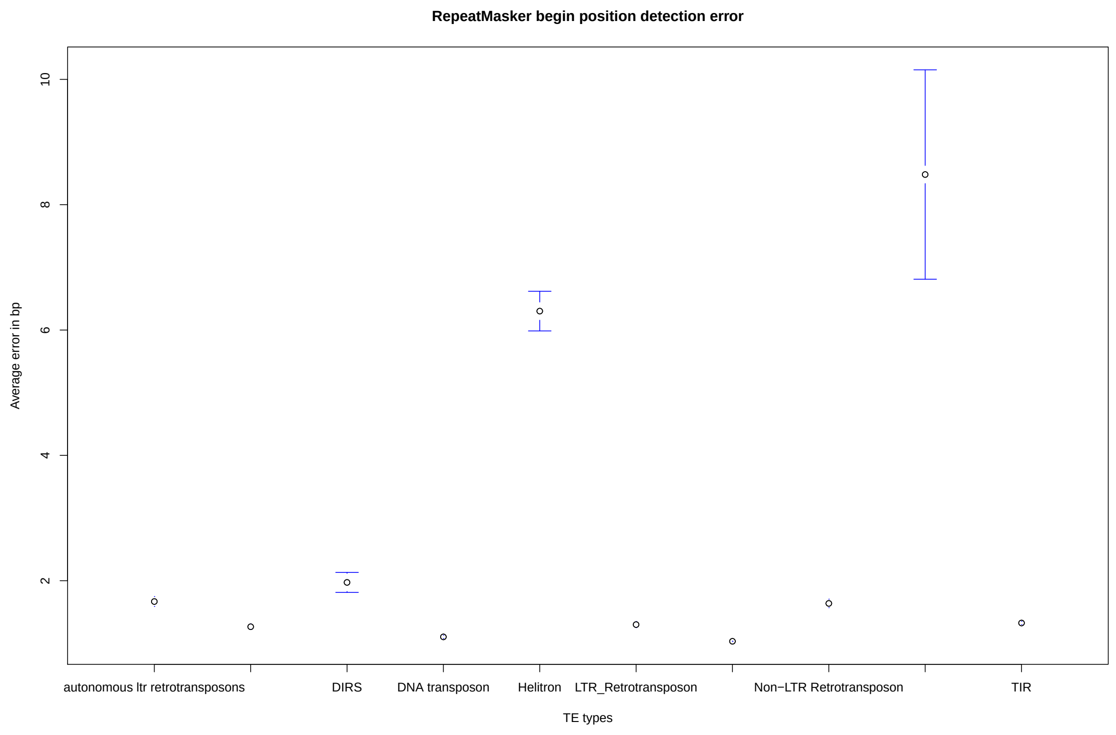

Figure 4.56: RepeatMasker "begin error"

Figure 4.59 shows the average error in TE length detection according to the actual TE length. There is no statistical evidence that the TE length is related with the tool's performance in detecting the actual length, as the correlation coefficient shows: -0.14. Despite this, the figure shows that the standard deviation of the TE length error increases as the TE length increases.

Figure 4.60 shows the average error in TE length detection according to the mutation percentage in the sequences. The figure shows that as the mutation percentage increases, the error in TE detection increases. This fact is not corroborated by the correlation coefficient value of 0.

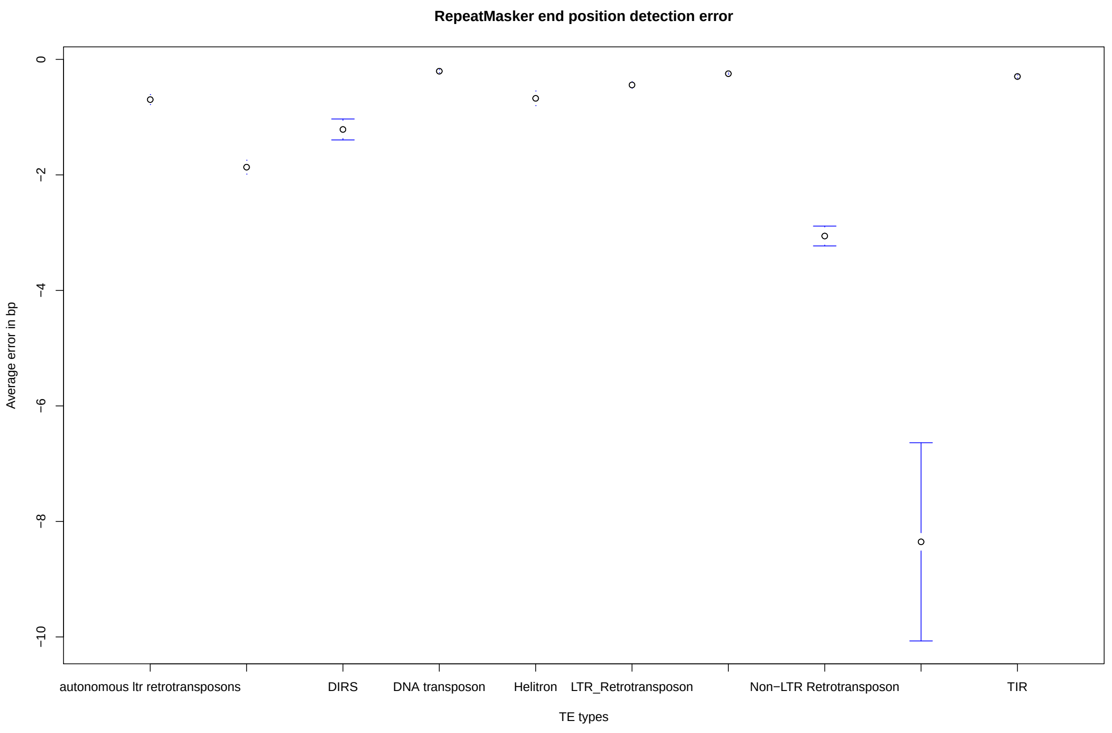

Figure 4.57: RepeatMasker "end error"

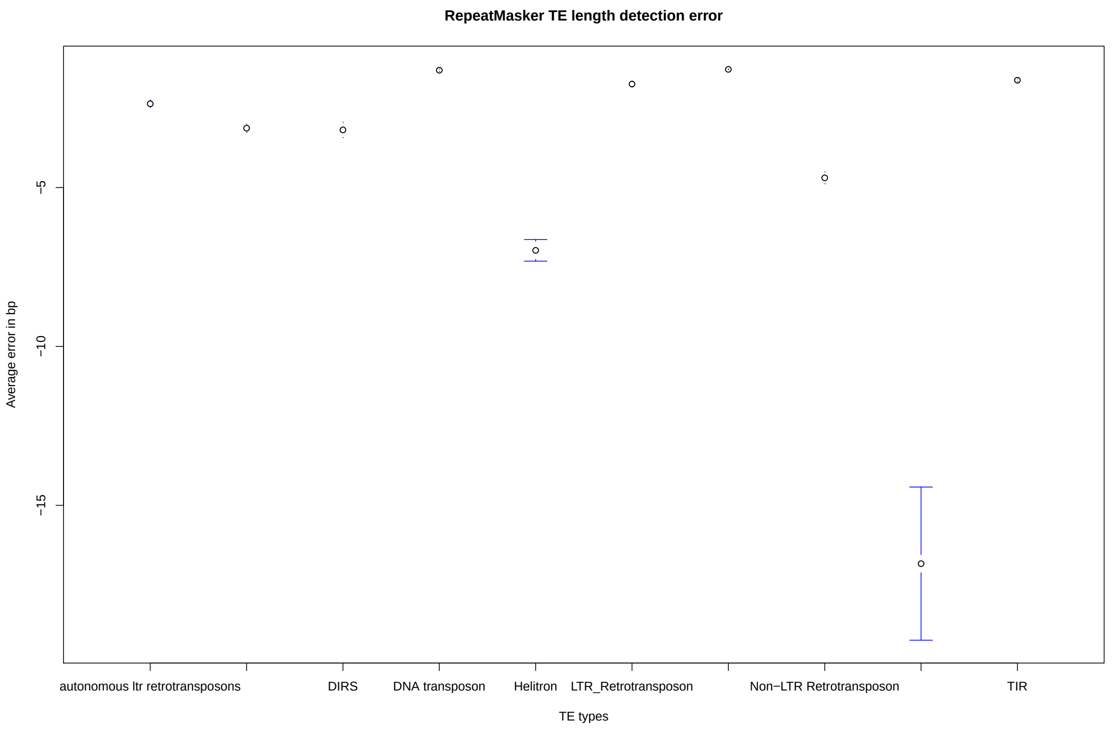

Figure 4.58: RepeatMasker TE "length error"

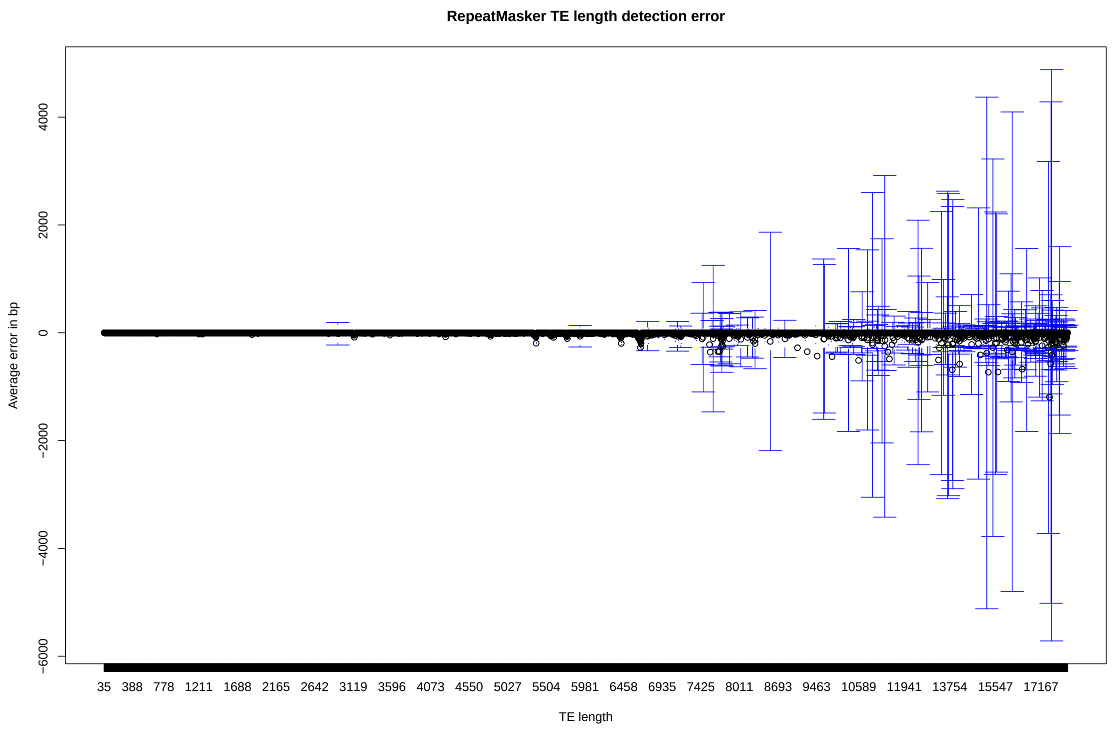

Figure 4.59: RepeatMasker's TE length error

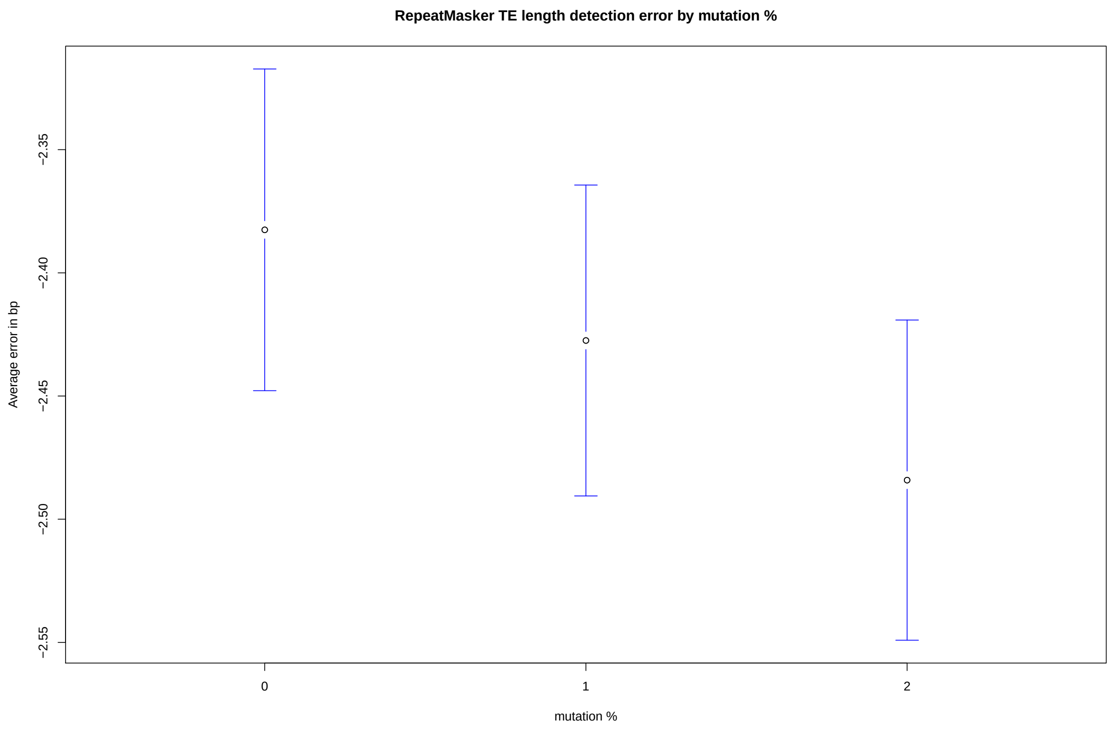

Figure 4.60: RepeatMasker's TE length error

4.2.7 Results Comparison

The results of all the five tools are here briefly compared and discussed. Figure 4.61 show the internal efficiency of all the tools. It suggests that LTR Finder is by far the most inefficient tool of all, having the lowest percentage of correctness. On the other hand, PILER and Censor tools were the most correct and were the ones that generated less false positive TEs.

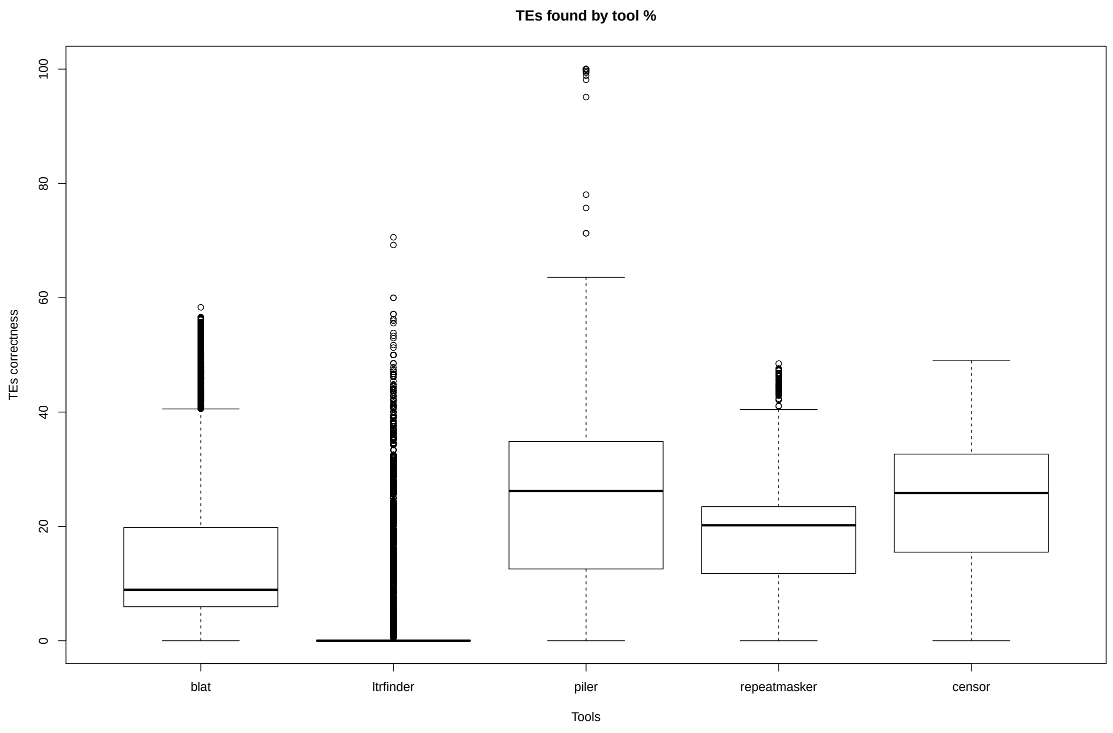

Figure 4.61: Correct TEs found by tools

Figure 4.62 shows the actual correctness of all the tools. LTR Finder had poor results with an average efficiency bellow 1%. The higher correctness of all the tools was achieved by Censor with an average efficiency above 70%. RepeatMasker had also interesting results, having it's average efficiency at roughly 50%.

In table 4.5 the average accuracy results of each tool regarding each TE type is presented.

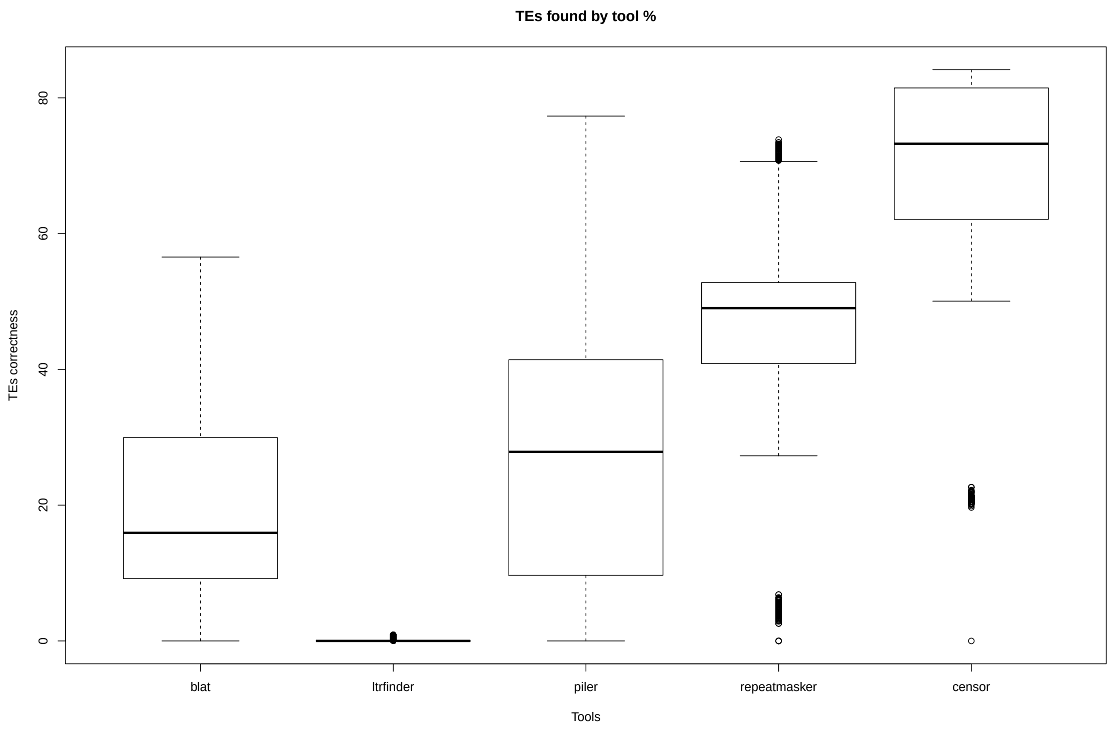

Figure 4.62: Corrrect TEs found by tools

Tool	**Aut. LTR Retrotransposons**	**Non Aut. LTR Retrotransposons**	**DIRS**	**Non-LTR Retrotransposons**	**Aut. Non-LTR Retrotransposons**	**Non Aut. Non-LTR Retrotransposons**	**TIRS**	**DNA TEs**	**Helitrons**	**Polintons**	**All**
BLAT	26.69%	18.12%	2.72%	23.12%	19.68%	37.69%	20.46%	13.67%	21.92%	11.14%	19.61%
Censor	61.49%	82.68%	81.43%	71.1%	74.02%	68.45%	78.85%	82.9%	52.13%	20.86%	67.38%
LTR Finder	0.17%	0.22%	0.1%	0.02%	0%	0%	0%	0%	0%	0%	0.05%
PILER	0.51%	38.05%	36.33%	37.46%	46.6%	10.24%	28.27%	25.63%	41.94%	23.56%	28.66%
Repeat Masker	51.66%	58.1%	31.1%	43.88%	51.71%	55.63%	57.66%	57.14%	42.23%	4.62%	45.28%

Table 4.5: Accuracy per TE type

It is clear that both Censor and RepeatMasker are the most accurate tools in finding TE elements. In the Polinton category the tool that has the higher efficiency is the PILER tool. LTR Finder has terrible results in finding any kind of TE.

4.3 Conclusion

This chapter clarified how the genome sequence dataset was generated and how the assessment process was planned. It followed by presenting the obtained results of the tools' assessment and conclusions where taken regarding their performance and behavior in different scenarios.

Chapter 5

Using Machine Learning to improve TE detection accuracy

In this chapter we describe the Machine Learning approaches used to try to improve the TE detection accuracy. In Section 5.1 it is described how Machine Learning methodologies are used in the aim of this work. In Sections 5.2 and 5.3 the models and the results are presented.

In Section 5.4 we summarize the conclusions from this chapter.

5.1 Model generation and Evaluation of the Results

Based on table 4.4 we tried to answer two main questions regarding the TE detection. The first question we tried to answer was what is/are the best tools to identify a TE with some given characteristics. The second question we want to answer is if a given sequence with some known characteristics is a TE element or not? And if it is a TE, which tool minimizes its detection error.

The *Rapidminer* software [Ri12] which uses *Weka* [oW] algorithm implementations was used to build models. This tool has a set of resources for data mining tasks, namely pre-processing, classification, regression, clustering, association rules, and visualization components.

We tried to apply discretization to the TE length variable. The distribution of this variable is represented in Figure 5.1.

We applied the Equal-depth Binning algorithm to discretize this variable in 5, 10, 20, 50 and 100 categories. T-test was used to assess if the differences in accuracy were statistically different.

The algorithms tested were a *Weka* Neural network implementation using 500 training cycles and 0.3 of learning rate; a *Weka* Bayes Network learning implementation; a *Weka* Random Forest classifier to build forest of random trees; a *Weka* implementation for generating Decision Trees based on the C4.5 algorithm.

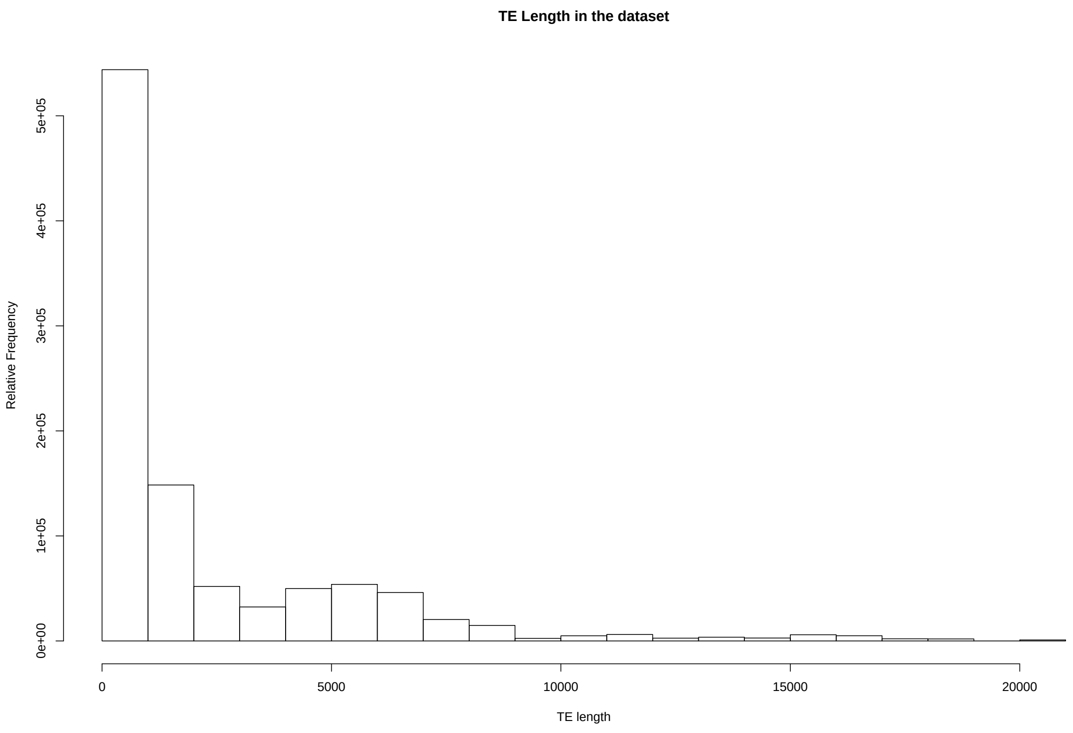

Figure 5.1: TE Length histogram

We tested different model generation algorithms, all subjected to a 10 fold cross-validation process, to assess their performance. The accuracy values present the average values observed in cross-validation.

5.2 Best tools for TE discovery

We tackled the problem of detecting the best tool for TE discovery as a classification problem. Our aim is to classify if a TE with some characteristics is detectable by one or more of the five tested TE detection tools. To do this, we created an artificial parameter which represents the tools that identified a TE. This parameter was generated using the data of the detection errors of the different tools. If a tool discovers a TE with a small or no error, then we accept it as a tool that is able to discover that TE.

As such we defined a boolean parameter for each tool which defines if a tool was able or not to detect that TE element. Using a concatenation of these parameters, we defined our label parameter which we called FOUNDTOOL, which is defined by the following expression.

FOUNDTOOL=BLATFOUND*10000+CENSORFOUND*1000+LTRFINDERFOUND*100+PILERFOUND*10+RepeatMaskerFOUND

We used different classification algorithms to generate plausible models, using a set of 201645 examples of TE elements equally distributed between the different TE classes. The features used as the input for the models were the TE length, the TE type and the FOUNDTOOL feature, as the aim to label.

Regarding the TE length discretization, we compared the performance of the best approach for this problem (Weka W-J48 algorithm) with the different numbers. The results are in Table 5.1.

Number of Categories	**Accuracy**	**Tree Length**
No discretization	67.39%	1182
5	45.33%	42
10	47.02%	73
20	48.19%	146
50	51.28%	419
100	53.44%	838

Table 5.1: Accuracy and complexity for different discretization values for TE length

According to the values obtained, and presented in Table 5.1, the best approach would be to use TE length undiscretized since the accuracy differences are significant for a confidence of 95%. But using this the outcome decision tree would be very large. Thus we decided to use this variable discretized in 50 categories to have a smaller resultant decision tree.

Algorithm	**Accuracy**
Neural Network	43.93%
Naive Bayes Net	48.88%
Random Forest	51.25%
Decision Trees	51.28%

Table 5.2: Classification algorithms model comparison

The best model (see Table 5.2) was generated by the Weka W-J48 algorithm, which is based on the C4.5 algorithm. The confusion matrix is displayed in the appendix Table C.2.

The results are far from satisfactory if one wants to know exactly which tools find a particular TE. However, if we widen our criteria and define the aim as finding at least one tool that finds that TE and not finding any other tool that does not find that TE, them the results are the ones in the appendix Table C.1. The overall accuracy of this model jumped from 51.28% to 57.17%.

Regarding these results we can conclude that the generated model can fairly indicate at least one tool that is capable of detecting a given TE suspicion.

5.3 Classifying an element as TE and finding the best TE detection tool

Given hypothetical predicted TE we want to know if it is either a real TE or not. And, if it is classified as a TE, we want to know which is the best tool from the five tested that minimizes the detection error. In Figure 5.2 the pipeline to solve this problem is represented.

The input for the pipeline is a predicted TE with information regarding its length, type, position and a FASTA file containing this suspicion. The output of the pipeline is the classification of the TE predicted as being TE or not. If it is classified as a TE it then it also provides the best tool for finding that TE, minimizing the errors of detection (either in start position, end position and length).

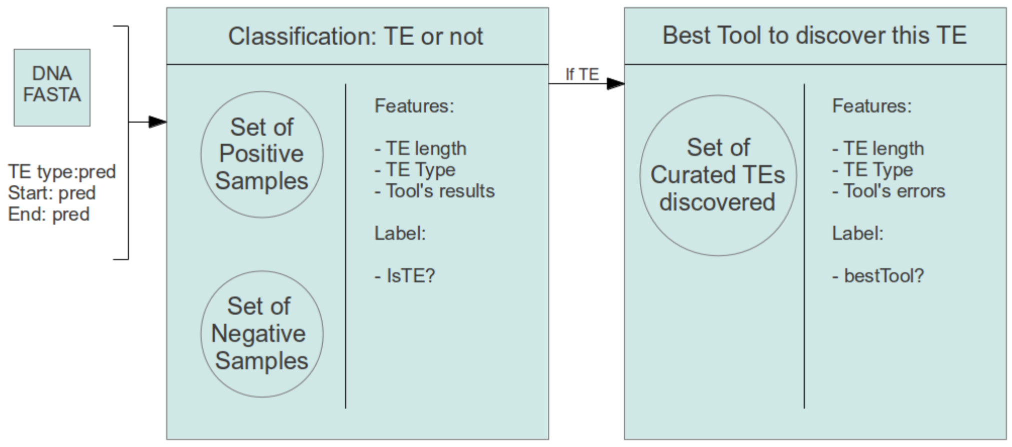

Figure 5.2: Pipeline scheme

5.3.1 Classify an element as TE

Classifying a potential TE candidate as a TE or not is a typical classification problem. In these terms, we used a dataset containing 325000 examples, equally distributed in terms of TE types and in terms of being real TEs or false positives. The features used as the input for the models were the discretized TE length, the TE type, the FOUNDTOOL and a IS_TE feature as the class. The IS_TE feature is a boolean which indicates whether a given example is or is not a TE.

We evaluated the impact of discretizing the TE length feature. The results are in Table 5.3.

Number of Categories	**Accuracy**	**Number of Leaves**
No discretization	98.92%	677
5	94.48%	261
10	95.36%	391
20	96.32%	613
50	97.09%	1086
100	97.78%	1921

Table 5.3: Accuracy and complexity for different discretization values for TE length

Again, not using discretization provides the best results since the difference of accuracies is statistically significant for 95% confidence.

We tested different model generation algorithms, all subjected to a 10 fold cross-validation process, to assess their performance. In Table 5.4 the different approaches used are compared.

Algorithm	**Accuracy**
Neural Network	69.01%
Naive Bayes Net	96.30%
Random Forest	98.90%
Decision Trees	98.92%

Table 5.4: Classification algorithms model comparison

Given these results, the best model was achieved with Decision Trees, which was the algorithm that scored higher on the accuracy. In Table 5.5 it is presented the confusion matrix of the Decision Tree model.

	True TE	**False TE**	**Class Prediction**
Predicted TE	140850	1542	98.92%
Predicted no TE	1982	181021	98.92%
Class Recall	98.61%	99.16%	

Table 5.5: Confusion matrix for Decision Trees in TE classification

Given the results obtained we can conclude that this classifier has an high accuracy and can perform really well with the tested artificial data. It can inform whether a TE prediction is correct or not.

5.3.2 Finding the best TE detection tool

The next phase of this pipeline aims to predict which is the tool that, for a given TE candidate that was classified as a TE, discovers it with the minimum location error. To do this we used a set of about 142900 examples of TE elements equally distributed between the different TE classes. Using as features the TE length, the TE type, BLATERROR, CENSORERROR, LTRFINDERERROR, PILERERROR, REPEATMASKER ERROR and bestTool, the last one as the aim of labeling. The ERROR variables are the module of the sum of the errors in beginning, end and length of TEs in the respective tools. The bestTool feature is the name of the tool that has the minimum location error.

Regarding the TE length discretization, we compared the performance of the best approach for this problem (Decision Trees) with the different numbers. The results are in table 5.6.

Number of Categories	**Accuracy**	**Tree Length**
No discretization	83.18%	945
5	75.33%	11
10	75.33%	11
20	75.33%	11
50	76.28%	256
100	77.83%	470

Table 5.6: Comparative model results with different discretization values for TE length

Based on the results presented in Table 5.6 the best scenario would be to use TE length undiscretized since the accuracy difference are significant for 95% confidence. However this would result in a very large decision tree. Thus we decided to use the TE length dicretized in 100 categories.

We tested different model generation algorithms, all subjected to a 10 fold cross-validation process, to assess their performance. In Table 5.7 the results obtained with the different learner algorithms are compared.

Algorithm	**Accuracy**
Neural Network	72.61%
Naive Bayes Net	74.39%
Random Forest	77.78%
Decision Trees	77.83%

Table 5.7: Classification algorithms model comparison

The best model was produced using Decision Trees, which was the algorithm that scored higher on the accuracy. In Table 5.8 it is represented the confusion matrix of this model.

	True Censor	**True BLAT**	**True LTR Finder**	**True Repeat-Masker**	**True Piler**	**Class Predic-tion**
Predicted Censor	8357	2082	21	491	42	76.02%
Predicted BLAT	18285	101020	143	8433	1306	78.20%
Predicted LTR Finder	0	0	0	0	0	0.00%
Predicted RepeatMasker	223	531	0	1361	25	63.60%
Predicted Piler	4	81	0	0	427	83.40%
Class Recall	31.10%	97.40%	0.00%	13.23%	23.72%	

Table 5.8: Confusion matrix for Decision Trees in the classification of best tool

Given the results obtained we can conclude that this classifier has an high accuracy and can perform well with the tested artificial data. It is also worth to mention that the LTR Finder tool was never used in this context as the location error performance of this tools is much lower than the others.

5.4 Conclusion

This chapter showed that applying machine learning models in the TE detection scope can further improve the accuracy of TE detection and annotation. In all the different problems, the approach that produced best results was Decision Trees (W-J48 Weka implementation). We decided in the last two approaches to lose some accuracy in the models by discretizing the TE length variable in 50 and 100 classes. We lost some accuracy but gained less complex models.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this master thesis one of the main goals was to make a detailed assessment of existent TE detection tools and compare their performances in different scenarios.

The results obtained suggest that both Censor and RepeatMasker are the the first and second most accurate tools in finding TE elements, respectively. In a particular category, in the Polinton TEs, the tool that had the best performance was the PILER tool. In an opposite position is the LTR Finder tool which has achieved, by far, the worse results in this comparison with very low TE detection accuracies.

BLAT and RepeatMasker had some problems detecting DIR TEs. On the other hand, Censor scored exceptionally well in this TE category. Polinton TEs were also a problem for tools like RepeatMasker, Censor and BLAT. In this case, PILER performed especially well, outscoring all the other tools.

In terms of TE boundaries errors, except for the LTR Finder performance, all the tools performed acceptably well. The biggest issues occurred on the detection of the boundaries of Polinton TEs and Piler had some trouble in detecting DIR TEs.

The sensitivity of these tools for mutations present in the sequences analyzed was also a theme we wanted to clarify. Given the results obtained we can conclude that BLAT and Piler tools are influenced by the amount of noise present in the DNA sequences. On the other hand, Censor, LTR Finder and RepeatMasker were capable of handling mutations on the sequences analyzed without dropping significantly their performances.

A key goal in this dissertation was to provide help for researchers in the TE detection using machine learning techniques was.

Using the data from the tools' predictions over our dataset, we generated a classification model using Decision Trees C4.5 algorithm implementation from *Weka* that provided an answer to what tool, or tools, are capable of discovering a TE prediction. The accuracy of this model was 51.28% for detecting using all tools. However if we want at least one tool capable of detecting a TE suspicion, then the accuracy of this model jumps to 57.17%. Despite not being a high accuracy, it can be used as a fairly accurate indicator to which tool to apply to a TE candidate.

We have also generated a pipeline consisting of two models that predicted if a given TE candidate is a TE or not and if it was a TE, it would tell which tool to use to minimize the boundaries error of that TE.

The first model of this pipeline, which classifies a TE candidate as a TE or not was created using the C4.5 decision tree algorithm. The accuracy for this model is very high when evaluated with our artificial data, over 98%. This means that there is potential in this approach for TE detection. This model combines the different tool's opinions over the TE candidate, the type of TE that the researcher thinks it is and the approximate length to provide an answer.

The second model of this pipeline uses the TE classified by the previous model and the estimated errors of the different tools to tell which tool should be used to detect this TE with minimal error in the boundaries. The accuracy of this model over our artificial data was 76.28%, suggesting that this is a valid approach that researchers can use to reduce TE detection errors.

All in all, although further research should be done regarding the usage of machine learning models for TE detection improvement, we present clear evidence that this approach can further advance the TE detection development.

6.2 Future Work

Having analyzed these TE detection results and modeled machine learning classifiers to answer important questions regarding TE detection, many future lines of work open.

One of the main focuses on future work could be to evaluate the generated models with real data from living beings genomes. Although the results are good for artificial generated DNA sequences, there is lack of evidence that these models can work with other genomes. They are a proof that using machine learning over the results of TE detection can give further information for researches though they still lack the validation needed to become an accepted approach in this context.

Detailing even further the tools' assessment could also lead to use of better features for TE detection classification and discover the best tool to find TEs. Namely, if one would use the type feature with more detailed categories (in this work we have defined to group TEs in ten different categories), the overall performance of the models generated could improve.

Another interesting path to take would be to develop an interface for the pipeline used in this work. The researcher could visualize a genome, select the boundaries of a sequence and give his opinion. This input would be processed by the pipeline which would tell if the sequence selected is or is not a TE. If it was a TE then the interface would run the best tool to discover the boundaries of that TE and show the researcher the outcome.

Machine learning techniques could be applied not only on statistical data from TE detection but also on researchers opinions on whether a sequence is a TE or not. Using Inductive Logic Programming (ILP) one could combine the statistical facts regarding TE detection with the knowledge and experience of researchers in this area so the final outcome would be a decision based on the learning from researchers and from TE detection tools results.

Appendix A

FASTA format

According to [Lab04] [Ins] [Inf07] in this format each sequence consists of a single header line started with ">" providing the sequence name and, optionally, a description followed by lines of sequence data. There should be no space between the ">" and the first letter of the identifier and usually each line is formated to 60 characters long and should be no longer than 80 characters. Lines started with a semicolon are ignored as they are treated as comments. Bellow is an example of a DNA sequence in FASTA format.

```
    >Name and description
GTCCTGACTTTCCTCGCTTGGTGGTTTGAGTGGACCTCACAGGCCAGTGCCGGGCCCCTCATAGGAGAGG
AAGCTCGGGAGGTGGCCAGGCGGCAGGAAGGCGCACCCCCCCAGCAATCCGCGCGCCGGGACAGAATG
CTGCAGGAACTTCTTCTGGAAGACCTTCTCCTCCTGCAAATAAAACCTCACCCATGAATGCTCACGCAAG
TTTAATTACAGACCTGAA
```

In FASTA files containing multiple sequences, one sequence ends when a ">" appears to mark the start of another one. In the scope of this dissertation, the sequences are filled with combinations of four letters representing the four nucleotids present in the DNA sequences:

- A - Adenine
- C - Guanine
- G - Cytosine
- T - Thymine

FASTA format

Appendix B

Transposon Detection Tools

In this chapter the TEs detection tools used in this work are detailed in terms of input parameters, software requirements, used techniques and generated outputs.

B.1 BLAT

BLAT is an alignment tool that is more accurate and 500 times faster than popular existing tools for mRNA/DNA alignments and 50 times faster for protein alignments at sensitivity settings typically used when comparing vertebrate sequences.

The main steps of this tool are:

1. First it indexes of all nonoverlapping K-mers in the genome and fits it inside the RAM
2. The index is used to find regions in the genome likely to be homologous to the query sequence.
3. It stitches together these aligned regions into larger alignments, typically genes.
4. It revisits small internal exons possibly missed at the first stage and adjusts large gap boundaries that have canonical splice sites where feasible.

The input for this tool is a FASTA file containing the sequence to be analyzed and a library file containing TEs in FASTA format. In this context we have used the Repbase library [Rep].

The standard output of the BLAT tool is a .psl file containing the information of the hits. This output is them processed by a perl script which converts it into a standar GFF file.

The parameters used to run this tool were the following:

- -out: This parameter defines the output format. We have chosen to have the output in .psl, which is a tab separated format.
- -t: This parameter defines the database type. For this work we have chosen DNA type since we are dealing with DNA sequences.

- -q: This parameter defines the query type. For this work we have chosen DNA type since we are dealing with DNA sequences.
- -tileSize: This parameter sets the size of match that triggers an alignment. The value is the default value, which is 11 for DNA sequences.
- -stepSize: This parameter sets the spacing between tiles. The value is by default value, which is 11 for DNA sequences.
- -oneOff: This parameter if set to 1 allows one mismatch in a tile and still triggers an alignment. We have used the default value, which is 0, to not allow any mismatch.
- -minMatch: This parameter sets the number of tile matches. We have used the default value, which is 2 for nucleotides.
- -minScore: This parameter sets the minimum score. We have used the default value.
- -minIdentity: This parameter sets the minimum sequence identity, in percentage. We have used the default value, which is 90% for nucleotide searches.
- -maxGap: This parameter sets the size of maximum gap between tiles in a clump. We have used the default value, which is 2.
- -repMatch: This parameter sets the number of repetitions of a tile allowed before it is marked as overused. We have used the default value, which is 1024.
- -maxIntron: This parameter sets maximum intron size. We have used the default value, which is 750000.

B.2 Censor

CENSOR is a tool that identifies repetitive elements in genomic sequences. to have query sequences aligned against a reference collection of repeats.

The main steps of this tool are:

1. CENSOR uses a BLAST approach to search the query sequence against a database of TEs.
2. After it uses an information theoretic method to detect simple sequence repeats such as satellite DNA and low complexity sequences.
3. It post-processes data to give an interactive positional map of the query sequence and calculates the similarity values and positive score values for alignments between query and element consensus sequences.

Censor searches the query sequence against a library of repetitive elements using one the following third-party tools:

- NCBI BLAST - http://www.ncbi.nlm.nih.gov
- WU-BLAST - http://blast.advbiocomp.com/

In this work we have opted for the use of NCBI BLAST as this is a free tool which fulfills all our requirements. As for WU-BLAST, since it is a proprietary tool, it can't be used in the scope of this work.

The input for this tool is a FASTA file containing the sequence to be analyzed and a library file containing TEs. In this context we have used the Repbase library [Rep].

The output of CENSOR tool includes a map of found matches with coordinates of corresponding fragments in database and query. If no matches were found no map will be produced. This file is post-processed using a Perl script to create a GFF file containing the information of the CENSOR's hits.

The parameters used to run this tool were the following:

- -s: This parameter sets sensitivity mode to sensitive.

B.3 LTR Finder

LTR Finder is an efficient program for finding full-length LTR retrotranspsons in genome sequences.

The main steps of this tool are:

1. It constructs all exact match pairs by a suffix-array based algorithm and extends them to long highly similar pairs.
2. It uses Smith-Waterman algorithm to adjust the ends of LTR pair candidates to get alignment boundaries.
3. These boundaries are subject to re-adjustment using supporting information and reliable LTR TEs are selected.
4. It tries to identify PBS, PPT and RT inside LTR pairs by build-in aligning and counting modules. RT identification includes a dynamic programming to process frame shift.
5. At last, this tool reports possible LTR retrotransposon models in different confidence levels according to how many signals and domains they hit.

The input for this tool is a FASTA file containing the sequence to be analyzed.

The output of LTR Finder tool is a file containing a summary output of the results. This file is post-processed using a Perl script to create a GFF file containing the information of the LTR Finders' hits.

The parameters used to run this tool were the following:

- -w: This parameter sets the output mode. The value selected was 2, to have the summary output.

B.4 PILER

PILER, which stand for Parsimonious Inference of a Library of Elementary Repeats, is a tool designed to search for repetitive elements in genome sequences. In the scope of this work we have used the TR search module, which finds similar pairs that may be terminal repeats in mobile elements such as the LTR superfamilies or Tc1.

The main steps of this module are:

1. In the first phase searches for candidate pairs of terminal repeats.
2. After this, it searches for pairs of candidates that confirm each other. Two pairs confirm each other if the terminal repeats from both pairs align globally to each other and the spacing between them is similar. Confirming pairs are recognized by finding hits that align one repeat from each pair.
3. Finally it finds families of confirming pairs.

PILER uses the following third-party tools:

- PALS - http://www.drive5.com/pals
- MUSCLE - http://www.drive5.com/muscle

PALS is used to find local alignments of a genome to itself while MUSCLE is used to create multiple alignments of each repeat family.

The input for this tool is a FASTA file containing the sequence to be analyzed.

The output of PILER is a GFF file containing the information of the LTR Finders' hits.

The parameters used to run this tool were the following:

- -mintrlength: This parameter defines the minimum T value in the first phase. We used the default value, 50.
- -maxtrlength: This parameter defines the maximum T value in the first phase. We used the default value, 2000.
- -mintrspacing: This parameter defines the minimum S value in the first phase. We used the default value, 50.
- -maxtrspacing: This parameter defines the minimum S value in the first phase. We used the default value, 12000.
- -minspacingratio: This parameter defines the minimum value of the spacing ratio. We used the default value, 0.9.
- -minhitratio: This parameter defines the minimum value for the hit ratio. We used the default value, 0.5.

- -mindistpairs: This parameter defines the minimum D value for the second phase. We used the default value, 50000.
- -minfam: This parameter defines the minimum family size for the last phase. We used the default value, 3.

B.5 RepeatMasker

RepeatMasker is a tool that searches DNA sequences for interspersed repeats and low complexity DNA sequences.

The main steps of this module are:

1. RepeatMasker scans the query sequence using a BLAST like tool against the library of consensus sequences provided.
2. A score matrix is first constructed based on exact word matches between the library sequences and the query sequence. This is then expanded to include a band of sequences that surround the exact match.
3. Since there can be many consensus sequences in the library of consensus sequences that match the same region of the query sequence, the search engines return the matrices that have less than 80% to 90% overlap with each other. The sequence with the highest SW score is selected for annotation after various approximation improvements.

RepeatMasker searches the query sequence against a library of repetitive elements using one the following third-party tools:

- RMBLAST - www.repeatmasker.org
- WU-BLAST - http://blast.advbiocomp.com/

RMBLAST is a RepeatMasker compatible version of the standard NCBI BLAST suite. In this work we have opted for the use of RMBLAST as this is a free tool which fullfils all our requirements. As for WU-BLAST, since it is a proprietary tool, it can't be used in the scope of this work.

The input for this tool is a FASTA file containing the sequence to be analyzed and a library file containing TEs. In this context we have used the Repbase library [Rep].

The output of the RepeatMasker tool is a detailed annotation of the repeats that are present in the query sequence. The file is a GFF file and therefore there is no post-processing.

The parameters used to run this tool were the following:

- -lib: This parameter sets the costum library of consensus to be used. We used the Repbase library.

- -cutoff: This parameter sets the cutoff score for masking repeats when using the -lib option. We used the 250 value, since according to the RepeatMasker documentation, this value will guarantee that all matches are real.
- -gff: This parameter sets the output format to GFF.
- -q: This parameter sets the search for the quick mode, which is 5% to 10% less sensitive and 3 to 4 times faster than default. This option was taken due to time constrains.
- -pa: This parameter set the number of processors to use in parallel. We used the value 4.
- -no_is: This parameter, when selected, makes the tool skips bacterial insertion element check.
- -norna: This parameter makes RepeatMasker ignore some pseudogenes and small tRNAs and SnRNAs which can be mistaken by SINEs due to their close similarity.

Appendix C

Confusion Matrices

	Censor	Blat, Censor	Blat	None	Blat, Censor, Piler	Blat, LTR Finder	Blat, Piler	Blat, LTR Finder, Piler	Blat, Censor, RepeatMasker	Blat, RepeatMasker	RepeatMasker	Blat, Censor, Piler, RepeatMasker	Censor, RepeatMasker	Blat, LTR Finder, RepeatMasker	Blat, Censor, LTR Finder, RepeatMasker	Blat, Piler, RepeatMasker	Censor, Piler, RepeatMasker	Piler, RepeatMasker	Blat, LTR Finder, Piler, RepeatMasker	Censor, Piler	Piler	Censor, LTR Finder, Piler, RepeatMasker	LTR Finder, RepeatMasker	All	Censor, LTR Finder	LTR Finder	Censor, LTR Finder, RepeatMasker	Censor, LTR Finder, Piler	Blat, Censor, LTR Finder	LTR Finder, Piler, RepeatMasker	Class Prediction
Censor	307	44	71	133	86	0	99	0	3	48	119	0	9	0	0	73	1	19	0	157	17	0	0	0	0	0	0	0	0	0	25.89%
Blat, Censor	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00%
Blat	2	29	63	13	0	3	12	0	32	74	9	1	7	2	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	25.40%
None	160	76	192	1299	92	0	391	0	167	227	517	212	323	0	0	303	53	153	0	9	487	0	0	0	0	0	0	0	0	0	27.87%
Blat, Censor, Piler	39	66	1	4	364	0	1	0	21	8	23	83	24	0	0	27	33	4	0	13	0	0	0	0	0	0	0	0	0	0	51.20%
Blat, LTR Finder	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00%
Blat, Piler	1	0	23	27	1	0	115	0	23	18	25	98	27	0	0	105	34	35	0	1	24	0	0	0	0	0	0	0	0	0	20.65%
Blat, LTR Finder, Piler	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00%
Blat, Censor, RepeatMasker	504	1399	502	735	176	0	7	0	39516	7420	2650	916	8985	0	1	415	118	204	0	39	72	0	0	0	1	0	0	1	0	0	62.07%
Blat, RepeatMasker	79	590	1230	380	12	8	21	3	2915	6297	1326	23	1160	27	13	74	1	3	3	0	0	0	6	0	0	2	2	0	2	0	44.42%
RepeatMasker	45	16	48	207	30	0	75	0	853	610	1813	259	580	0	0	490	94	284	0	7	52	0	0	0	0	0	0	0	0	0	33.19%
Blat, Censor, Piler, RepeatMasker	647	570	315	347	1830	2	927	7	11649	2452	1427	41561	5056	43	3	7048	6898	1965	75	984	265	0	41	7	1	5	5	0	0	1	49.40%
Censor, RepeatMasker	187	139	15	18	37	0	32	2	926	610	93	376	2624	2	0	34	520	8	1	54	1	0	4	0	0	1	0	0	0	0	46.16%
Blat, LTR Finder, RepeatMasker	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00%
Blat, Censor, LTR Finder, RepeatMasker	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00%
Blat, Piler, RepeatMasker	53	43	37	58	100	0	164	0	1127	1260	1004	2170	825	1	7	4288	290	619	3	24	20	1	3	6	0	0	4	0	0	0	35.42%
Censor, Piler, RepeatMasker	138	1	0	2	26	0	0	0	86	13	9	654	1763	0	0	46	3920	12	0	234	1	14	0	0	3	0	8	3	0	0	56.54%
Piler, RepeatMasker	0	0	0	0	0	0	0	0	9	0	16	65	11	0	0	0	43	188	0	0	0	0	0	0	0	0	0	0	0	0	56.63%
Blat, LTR Finder, Piler, RepeatMasker	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00%
Censor, Piler	15	1	0	0	23	0	0	0	4	0	0	32	6	0	0	9	15	3	0	108	2	0	0	0	0	0	0	0	0	0	49.54%
Piler	3	0	0	568	0	0	62	0	0	0	0	0	0	0	0	0	0	0	0	0	943	0	0	0	0	0	0	0	0	0	59.84%
Censor, LTR Finder, Piler, RepeatMasker	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00%
LTR Finder, RepeatMasker	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00%
All	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00%
Censor, LTR Finder	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00%
LTR Finder	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00%
Censor, LTR Finder, RepeatMasker	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00%
Censor, LTR Finder, Piler	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00%
Blat, Censor, LTR Finder	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00%
LTR Finder, Piler, RepeatMasker	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00%
Class Recall	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00%

Table C.1: Confusion matrix of Decision Tree applyed to the Best Tools for TE discovery

	Censor	Blat, Censor	Blat	None	Blat, Censor, Piler	Blat, LTR Finder	Blat, Piler	Blat, LTR Finder, Piler	Blat, Censor, RepeatMasker	Blat, RepeatMasker	RepeatMasker	Blat, Censor, Piler, RepeatMasker	Censor, RepeatMasker	Blat, LTR Finder, RepeatMasker	Blat, Censor, LTR Finder, RepeatMasker	Blat, Piler, RepeatMasker	Censor, Piler, RepeatMasker	Piler, RepeatMasker	Blat, LTR Finder, Piler, RepeatMasker	Censor, Piler	Piler	Censor, LTR Finder, Piler, RepeatMasker	LTR Finder, RepeatMasker	All	Censor, LTR Finder	LTR Finder	Censor, LTR Finder, RepeatMasker	Censor, LTR Finder, Piler	Blat, Censor, LTR Finder	LTR Finder, Piler, RepeatMasker	Class Prediction
Censor	307	44	71	133	86	0	99	0	3	48	119	0	9	0	0	73	1	19	0	157	17	0	0	0	0	0	0	0	0	0	51.18%
Blat, Censor	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00%
Blat	2	29	63	13	0	3	12	0	32	74	9	1	7	2	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	87.10%
None	160	76	192	1299	92	0	391	0	167	227	517	212	323	0	0	303	53	153	0	9	487	0	0	0	0	0	0	0	0	0	27.87%
Blat, Censor, Piler	39	66	1	4	364	0	1	0	21	8	23	83	24	0	0	27	33	4	0	13	0	0	0	0	0	0	0	0	0	0	62.87%
Blat, LTR Finder	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00%
Blat, Piler	1	0	23	27	1	0	115	0	23	18	25	98	27	0	0	105	34	35	0	1	24	0	0	0	0	0	0	0	0	0	57.27%
Blat, LTR Finder, Piler	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00%
Blat, Censor, RepeatMasker	504	1399	502	735	176	0	7	0	39516	7420	2650	916	8985	0	1	415	118	204	0	39	72	0	0	0	1	0	0	1	0	0	62.07%
Blat, RepeatMasker	79	590	1230	380	12	8	21	3	2915	6297	1326	23	1160	27	13	74	1	3	3	0	0	0	6	0	0	2	2	0	2	0	65.97%
RepeatMasker	45	16	48	207	30	0	75	0	853	610	1813	259	580	0	0	490	94	284	0	7	52	0	0	0	0	0	0	0	0	0	91.21%
Blat, Censor, Piler, RepeatMasker	647	570	315	347	1830	2	927	7	11649	2452	1427	41561	5056	43	3	7048	6898	1965	75	984	265	0	41	7	1	5	5	0	0	1	49.41%
Censor, RepeatMasker	187	139	15	18	37	0	32	2	926	610	93	376	2624	2	0	34	520	8	1	54	1	0	4	0	0	1	0	0	0	0	78.22%
Blat, LTR Finder, RepeatMasker	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00%
Blat, Censor, LTR Finder, RepeatMasker	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00%
Blat, Piler, RepeatMasker	53	43	37	58	100	0	164	0	1127	1260	1004	2170	825	1	7	4288	290	619	3	24	20	1	3	6	0	0	4	0	0	0	53.42%
Censor, Piler, RepeatMasker	138	1	0	2	26	0	0	0	86	13	9	654	1763	0	0	46	3920	12	0	234	1	14	0	0	3	0	8	3	0	0	66.18%
Piler, RepeatMasker	0	0	0	0	0	0	0	0	9	0	16	65	11	0	0	0	43	188	0	0	0	0	0	0	0	0	0	0	0	0	89.16%
Blat, LTR Finder, Piler, RepeatMasker	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00%
Censor, Piler	15	1	0	0	23	0	0	0	4	0	0	32	6	0	0	9	15	3	0	108	2	0	0	0	0	0	0	0	0	0	81.65%
Piler	3	0	0	568	0	0	62	0	0	0	0	0	0	0	0	0	0	0	0	0	943	0	0	0	0	0	0	0	0	0	63.77%
Censor, LTR Finder, Piler, RepeatMasker	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00%
LTR Finder, RepeatMasker	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00%
All	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00%
Censor, LTR Finder	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00%
LTR Finder	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00%
Censor, LTR Finder, RepeatMasker	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00%
Censor, LTR Finder, Piler	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00%
Blat, Censor, LTR Finder	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00%
LTR Finder, Piler, RepeatMasker	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00%
Class Recall	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.00%

Table C.2: Confusion matrix of Decision Tree applyed to the Best Tools for TE discovery in which the accuracy is calculated based on the criteria: at least one tool that finds a TE and no tools that don’t find the TE

References

[Ala02] Weiner Alan M. SINEs and LINEs: the art of biting the hand that feeds you. *Current Opinion in Cell Biology*, 14(3):343–350, June 2002.

[Ame05] *The American Heritage Science Dictionary*. Houghton Mifflin Harcourt, 2005.

[BE02] Zhirong Bao and Sean R Eddy. Automated De Novo Identification of Repeat Sequence Families in Sequenced Genomes. *Genome Research*, 12(8):1269–1276, 2002.

[BHD08] Victoria P Belancio, Dale J Hedges, and Prescott Deininger. Mammalian non-LTR retrotransposons: For better or worse, in sickness and in health. *Genome Research*, 18(3):343–358, 2008.

[bio11] BioPerl, 2011.

[BJ93] Peter Buneman and Sushil Jajodia, editors. *Mining Association Rules between Sets of Items in Large Databases*, Washington, D.C., 1993.

[BQ07] Casey M Bergman and Hadi Quesneville. Discovering and detecting transposable elements in genome sequences. *Briefings in Bioinformatics*, 8(6):382–392, November 2007.

[CCGL02] N L Craig, R Craigie, M Gellert, and A M Lambowitz. *{M}obile {D}{N}{A} II*. ASM Press, 2002.

[CD92] Gregory F Cooper and Tom Dietterich. A Bayesian method for the induction of probabilistic networks from data. In *Machine Learning*, pages 309–347, 1992.

[CD03] M Joan Curcio and Keith M Derbyshire. The outs and ins of transposition: from Mu to Kangaroo. *Nat Rev Mol Cell Biol*, 4(11):865–877, November 2003.

[CP06] A Caspi and L Pachter. Identification of transposable elements using multiple alignments of related genomes. *Genome research*, 16(2):260–270, February 2006.

[EM05] Robert C Edgar and Eugene W Myers. PILER: identification and classification of genomic repeats. *Bioinformatics*, 21(suppl 1):i152–i158, 2005.

[FGG97] Nir Friedman, Dan Geiger, and Moises Goldszmidt. Bayesian Network Classifiers. *Machine Learning*, 29(2):131–163, 1997.

[Fly] FlyBase.

[FRI86] TREVOR J. FRIED, MIKE and WILLIAMS. Inverted Duplication-Transposition Event in Mammalian Cells at an Illegitimate Recombination Join. *MOLECULAR AND CELLULAR BIOLOGY*, 6(6):2179–2184, 1986.

[GP08] Graziano and Pesole. What is a gene? An updated operational definition. *Gene*, 417(1–2):1–4, 2008.

[Gyd] Gydb.

[HHD$^+$10] F Hormozdiari, I Hajirasouliha, P Dao, F Hach, D Yorukoglu, C Alkan, E E Eichler, and S C Sahinalp. Next-generation VariationHunter: Combinatorial algorithms for transposon insertion discovery. *Bioinformatics*, 26(12):i350–i357, 2010.

[HW10] Yujun Han and Susan R Wessler. MITE-Hunter: a program for discovering miniature inverted-repeat transposable elements from genomic sequences. *Nucleic Acids Research*, 2010.

[Inf07] National Center for Biotechnology Information. Blast Help: FASTA, 2007.

[Ins] European Bioinformatics Institute. About Nucleotide And Protein Sequence Formats.

[JKDP96] Jerzy Jurka, Paul Klonowski, Vadim Dagman, and Paul Pelton. Censor—a program for identification and elimination of repetitive elements from DNA sequences. *Computers &amp; Chemistry*, 20(1):119–121, 1996.

[Jud01] James M. DeLeo Judith E. Dayhoff. Artificial Neural Networks: Opening the Black Box. *Cancer*, 91(8), 2001.

[KBWRE09] Emily N Kroutter, Victoria P Belancio, Bradley J Wagstaff, and Astrid M Roy-Engel. The RNA Polymerase Dictates ORF1 Requirement and Timing of LINE and SINE Retrotransposition. *PLoS Genetics*, 5(4):15, 2009.

[Ken02] James J Kent. BLAT—The BLAST-Like Alignment Tool. *Genome Research*, 12(4):656–664, 2002.

[KJ08] Vladimir V Kapitonov and Jerzy Jurka. A universal classification of eukaryotic transposable elements implemented in Repbase. *Nat Rev Genet*, 9(5):411–412, May 2008.

[KR04] Alexey S Kondrashov and Igor B Rogozin. Context of deletions and insertions in human coding sequences. *Human Mutation*, 23(2):177–185, 2004.

[KUC$^+$11] Ryan Kennedy, Maria Unger, Scott Christley, Frank Collins, and Gregory Madey. An automated homology-based approach for identifying transposable elements. *BMC Bioinformatics*, 12(1):130, 2011.

[KVP$^+$04] Ruslan Kalendar, Carlos M Vicient, Ofer Peleg, Kesara Anamthawat-Jonsson, Alexander Bolshoy, and Alan H Schulman. Large retrotransposon derivatives: abundant, conserved but nonautonomous retroelements of barley and related genomes. *Genetics*, 166(3):1437–1450, 2004.

[Lab04] Göttingen Genomics Lab. FASTA format description, 2004.

[McC50] B McCLINTOCK. The origin and behavior of mutable loci in maize. *Proceedings of the National Academy of Sciences of the United States of America*, 36(6):344–355, June 1950.

[MDK99] J V Moran, R J DeBerardinis, and H H Kazazian. Exon shuffling by L1 retrotransposition. *Science*, 283(5407):1530–1534, 1999.

[MdR94] Stephen Muggleton and Luc de Raedt. Inductive Logic Programming: Theory and methods. *The Journal of Logic Programming*, 19–20, Supplement 1(0):629 – 679, 1994.

[Mit97] Tom Michael Mitchell. *Machine Learning*. McGraw-Hill Series in Computer Science. WCB/McGraw-Hill, Boston, MA, 1997.

[nat01] Initial sequencing and analysis of the human genome. *Nature*, 409(6822):860–921, February 2001.

[oW] University of Waikato. Weka.

[PBV02] Brooke D Peterson-Burch and Daniel F Voytas. Genes of the Pseudoviridae (Ty1/copia Retrotransposons). *Molecular Biology and Evolution*, 19(11):1832–1845, 2002.

[Pen07] E Pennisi. Genomics. DNA study forces rethink of what it means to be a gene. *Science (New York, N.Y.)*, 316(5831):1556–1557, June 2007.

[Pra08] Leslie A. Pray. Transposons: The Jumping Genes. *Nature Education*, 2008.

[Qui86] J R Quinlan. Induction of Decision Trees. *Mach. Learn.*, 1(1):81–106, 1986.

[Qui93] J Ross Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[Ref12] Genetics Home Reference. *Handbook: Help Me Understand Genetics*. Genetics Home Reference, 2012.

[Rep] Repbase.

[Ri12] Rapid-i.com. Rapidminer, 2012.

[RN03] Stuart J Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003.

[Rob] Arend Sidow Robert C. Edgar. George Asimenos, Serafim Batzoglou. Evolver.

[RT09] Mina Rho and Haixu Tang. MGEScan-non-LTR: computational identification and classification of autonomous non-LTR retrotransposons in eukaryotic genomes. *Nucleic Acids Research*, 37(21):e143, 2009.

[SBMP08] Surya Saha, Susan Bridges, Zenaida Magbanua, and Daniel Peterson. Computational Approaches and Tools Used in Identification of Dispersed Repetitive DNA Sequences. *Tropical Plant Biology*, 1(1):85–96, March 2008.

[Smi] Green P Smit AFA, Hubley R. RepeatMasker Open-3.0.

[VHS01] N Volfovsky, B J Haas, and S L Salzberg. A clustering method for repeat analysis in DNA sequences. *Genome Biol*, 2(8):RESEARCH0027+, 2001.

[WC53] J D Watson and F H C Crick. Molecular structure of nucleic acids: a structure for deoxyribose nucleic acid. *Nature*, 171(4356):737–738, 1953.

[WF05] Ian H Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann Publishers, San Francisco, CA, 2nd edition, 2005.

[WLBK01] Claus-Peter Witte, Quang Hien Le, Thomas Bureau, and Amar Kumar. Terminal-repeat retrotransposons in miniature (TRIM) are involved in restructuring plant genomes. *Proceedings of the National Academy of Sciences*, 98(24):13778–13783, 2001.

[WSHV$^+$07] Thomas Wicker, Francois Sabot, Aurelie Hua-Van, Jeffrey L Bennetzen, Pierre Capy, Boulos Chalhoub, Andrew Flavell, Philippe Leroy, Michele Morgante, Olivier Panaud, Etienne Paux, Phillip SanMiguel, and Alan H Schulman. A unified classification system for eukaryotic transposable elements. *Nat Rev Genet*, 8(12):973–982, December 2007.

[XW07] Zhao Xu and Hao Wang. LTR_FINDER: an efficient tool for the prediction of full-length LTR retrotransposons. *Nucleic Acids Research*, 35(suppl 2):W265–W268, 2007.