U.PORTO
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Towards a Novel Pipeline for Microsatellite Screening in Population Genomics

Filipe da Costa Alves

Dissertação de Mestrado apresentada à
Faculdade de Ciências da Universidade do Porto em
Bioinformática e Biologia Computacional

2019

MSC

2.º CICLO

FCUP
2019

U.PORTO

Towards a Novel Pipeline for Microsatellite Screening in Population Genomics

Filipe da Costa Alves

FC

Towards a Novel Pipeline for Microsatellite Screening in Population Genomics

Filipe da Costa Alves
Mestrado em Bioinformática e Biologia Computacional
Departamento de Biologia | Departamento de Ciências de Computadores
2019

Orientador
António Muñoz Mérida, Investigador Principal, Bioinformática, CIBIO-InBIO

Coorientador
Miguel Areias, Professor Auxiliar Convidado, Departamento de Ciências de Computadores, Faculdade de Ciências da Universidade do Porto

U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, _____/_____/_____

Abstract

Analysis of intra- and inter-population diversity has become important for defining the genetic status and distribution patterns of a species and a powerful tool for conservation programs, since high levels of inbreeding could lead into a whole population extinction in few generations. In this work, we combine next generation sequencing (NGS) with automatic computing to develop a genomic-oriented tool for characterizing microsatellites (SSR) at the population level. Herein, we describe a new Python pipeline, named Micro-Primers, designed to identify and design PCR primers for amplification of SSR loci from a multi-individual enriched microsatellite library. The pipeline takes as input just a FASTQ file containing sequences from NGS and returns a text file with information regarding the microsatellite markers, including number of alleles in the population, the melting temperature and the respective product of primer sets to easily guide the selection of optimal markers for the species.

A dataset belonging to bats of the species *Hiposideros vittatus* from two different population from Namibia and Botswana were used to evaluate its performance. Using a Intel-i7 Octacore processor and with 64 GB of memory, it took Micro-primers less than 120 seconds to execute the analysis, which would require not less than 24 hours to be complete if manual processes were to be employed. No differences were found in the results obtained via manual process and Micro-Primers.

The usage of this pipeline discards any kind of additional scripting and frees laboratories from agonizing manual analysis of thousands of possible microsatellite candidates for their projects, eliminating possible user bias while saving money and time.

Keywords : Bioinformatics, Next Generation Sequecing, Microssatellite, SSR, Pipeline, Population Genomics

Resumo

A análise da diversidade intra e inter-populacional tornou-se importante para definir a diversidade genético e padrões de distribuição de espécies e uma ferramenta poderosa para programas de conservação, uma vez que elevados níveis de consanguinidade podem levar á extinção de uma população inteira em poucas gerações. Neste trabalho, combinamos Sequenciação de Nova Geração (NGS) com computação automática para desenvolver uma ferramenta genomicamente orientada para a caracterização de microsatélites a nível populacional. Nesta perspectiva, descrevemos uma nova pipeline programada em Python, chamada Micro-Primers. Esta foi desenvolvida para identificar e desenhar iniciadores para PCR para a amplificação de loci de microssatélites a partir de uma biblioteca de microsatélites enriquecida, proveniente de múltiplos indivíduos. A pipeline tem como ficheiro de entrada apenas ficheiros em formato FASTQ contendo sequências provenientes de NGS e retorna um ficheiro de texto com informação a respeito dos marcadores de microssatélites, incluindo o número de alelos na população, temperatura de fusão, e respetivo tamanho dos conjuntos de iniciadores para facilmente guiar a seleção de marcadores ideais para as espécies.

Um conjunto de dados de morcegos da espécie *Hiposideros vittatus* pertencentes a duas populações diferentes da Namíbia e Botswana foram usados para avaliar o respetivo desempenho. Usando um processador Intel-i7 com 64 Gb de memória, o Micro-Primers demorou apenas cerca de 120 segundos para executar a análise, o que exigiria não menos de 24 horas para ser concluída se processos manuais fossem empregues. Nenhuma diferença foi encontrada entre os resultados obtidos pela pipeline e por processos manuais.

A utilização desta pipeline descarta a utilização de programas adicionais e liberta os laboratórios de morosas análises a milhares de possíveis microssatélites candidatos para os seus projectos, eliminando possível enviesamento do utilizador enquanto poupa recursos e tempo.

Palavras-Chave : Bioinformática, Sequenciação de Nova Geração, Microssatélites, Pipeline,

SSR, Genómica Populacional

Agradecimentos

Aos meus orientadores, Professor Miguel Areias e Professor António Mérida, pela ajuda, confiança e paciência. Sem os seus conhecimentos e disponibilidade não seria possível realizar este trabalho.

Aos meus colegas e amigos Luís Alves e Miguel Faria pelas aulas, trabalhos e gargalhas que partilhamos como primeiros alunos do mestrado.

À minha família escutista que desde de criança esteve sempre comigo e que me apoiou ao longo de todos este anos.

Aos meus grandes amigos das "Tentativas de Café" por nunca me perguntarem a pergunta proibída e por nunca cancelarem um café.

Aos meus pais, à Sardinha, aos meus irmãos e restante família pelo apoio incondicional, por tornarem possível esta minha caminhada e por estarem sempre presentes na minha vida.

Contents

List of Tables

List of Figures

Listings

Acronyms

A Adenine

BLAST Basic Local Alignment Search Tool

C Cytosine

cDNA Complementary Deoxyribonucleic Acid

ddNTP Di-deoxynucleotidetriphosphates

DNA Deoxyribonucleic Acid

dNTP Deoxynucleotidetriphosphates

ePCR Emulsion Polymerase Chain Reaction

FCUP Faculdade de Ciências da Universidade do Porto

G Guanine

GUI Graphic User Interface

HTML HyperText Markup Language

HTS High-Throughput Sequencing

ID Identifier

IUPAC-IUB Commission on Biochemical Nomenclature

LINE Long Interspersed Nuclear Elements

MAS Marker Assisted Selection

mRNA Messenger Ribonucleic Acid

NCBI National Center for Biotechnology Information

NGS Next Generation Sequencing

PCR Polymerase Chain Reaction

RAD-Seq Restriction Site Associated Sequencing

RNA Ribonucleic Acid

SBS Sequencing-by-Synthesis

SINE Short Interspersed Nuclear Elements

ssDNA single Stranded Deoxyribonucleic Acid

SSR Simple Sequence Repeat

T Thymine

Tm Melting Temperature

UTR Untranslated Region

Chapter 1

Introduction

1.1 Next Generation Sequencing

The first sequencing technology was described in 1977 and was named Sanger on behalf of its inventor. This technology dominated the genomic research from the 80's until 00's and still nowadays it is an important tool for getting the sequence of very complex and repeated regions or to close genome holes. Sanger sequencing consists on copying several times the same target DNA and infer the nucleotide that is added to the sequence at every single position. The major breakthrough was when chain-termination technique appeared to stop the DNA extension at random positions. The idea of this technique was to perform simultaneously the sequencing in 4 separated reactions with the same reagents, the DNA template to be sequenced, a DNA primer to start the amplification, a DNA polymerase to perform the DNA synthesis, and deoxynucleotidetriphosphates (dNTPs) to be added at the end of the new sequence. The only difference in the 4 reactions is the inclusion of modified di-deoxynucleotidetriphosphates (ddNTPs) that will not allow to continue the elongation of the sequence since they lack of the 3'OH group required for the formation of the joint with the next nucleotide. Every different reaction has a different di-deoxynucleotidetriphosphate (A, C, G or T) producing that in each reaction any of the produced sequences will end at this particular nucleotide but in different positions. The proportion of dNTP and ddNTP is crucial, since normal nucleotides of the same nitrogenous base than the di-deoxynucleotide are required to get strands of different lengths. If the proportion of ddNTPs is high all the strands will be prematurely stopped. Using a 4 lane gel electrophoresis it is possible to identify the nucleotide order present in the original strand [2] (Figure 1.1).

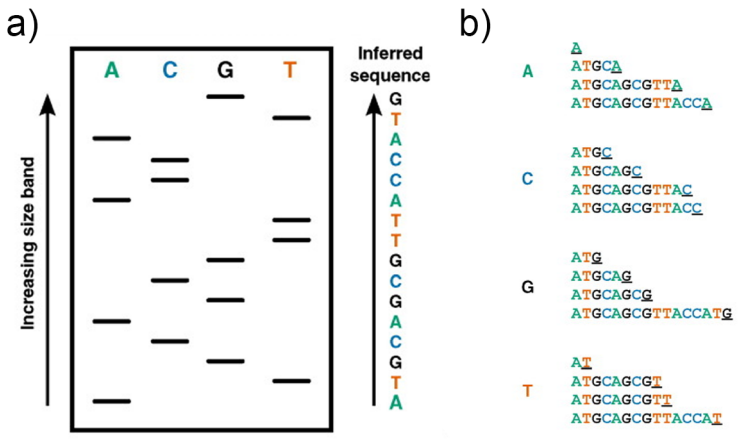

Figure 1.1: Illustration of a four lane electrophoresis containing fragments generated from Sanger sequencing. A example DNA is sequenced using Sanger's chain termination methodology, originating various reads with different lengths due to the use of ddNTPs, which stop chain amplification when added by DNA polymerase. All possible combinations are shown at (b). These fragments can be visualized using electrophoresis and the sequence is inferred by finding the lane in which the band is present for a given site since the ddNTP corresponds to the base in that position. Adapted from Heather *et al.* [1]

.

This technique was revolutionary, since it enabled the sequencing of DNA strands with high levels of accuracy, robustness and it is ease of use. A number of improvements were made to Fred Sanger's method, such as using fluorometric based detection allowing the use of one vessel instead of four [3], and improved detection through capillary based electrophoresis [4]. All this improvements lead to the creation of the first automated sequencing machines and protocols [1]. The evolution of this technique allowed scientist to obtain the first complete genomes of different species, such as *Escherichia coli* [5], *Arabidopsis thaliana* [6], *Drosophila melanogaster* [7] and most important, the completion of the human genome [8, 9].

Although Sanger method is still widely used, it has some disadvantages. Sanger sequencing has low sensitivity, being unable to detect mosaic alleles below a threshold of 15 to 20 %, it produces a low number of reads, important for studying somatic mutations. The cost for sequencing a whole genome is very expensive, and can't be used for parallel research of multiple targets [10, 11].

Together with the previous technologies developed, new technologies were created. In 2005 a new method was published called "Sequencing by Synthesis Technology" by 454 Life Sciences [12]. In this method a new strategy originated a great reduction of the necessary reaction volume and allowed the parallelization of the process producing at the same time a huge number of different sequences. This landmark paper started the now called Next Generation Sequencing (NGS) era.

NGS, also known as high-throughput sequencing (HTS), includes a number of methods, such as, template preparation, sequencing and imaging, and data analysis. Nowadays, there are three main competing technologies: Illumina/Solexa sequencing, Roche 454 sequencing and Ion sequencing from TermoFisher.

All NGS technologies need a library construction before the sequencing itself. A library is a collection of cloned DNA or cDNA fragments, originated from the DNA or RNA samples to be sequenced, joint to a known adapter, specific for the technology used. This adapter will be the link to the surface where the sequencing will be performed or even can be used as flag to identify different samples in the same sequencing run.

1.1.1 Illumina/Solexa

This technology was initially developed by Solexa which was after acquired by Illumina. It uses a method called bridge amplification. Instead of using a bead-based emulsion PCR (ePCR), a amplification reaction is performed on the base of a glass flow cell containing several lanes. This flow cell is covered with specific oligos, complementary to those added to the library in the preparation phase. The single stranded DNA (ssDNA) will bind to these oligos and bend in such a way that the ssDNA is connected to the two oligos simultaneously forming a bridge, hence the name. Priming begins at the priming binding site, located in the added adapters, and repeated cycles of denaturation and extensions occurs. This will originate the clonal amplification of all fragments along the flow cell, each composed of thousands of copies of a single fragment, resulting in forward and reverse copies of the same fragment [13]. Next, the strands are cleaved and the reverse strands are washed away, leaving only the forward strands, while the prime ends are blocked to avoid any unwanted priming. The sequencing phase starts with the extension of the primer sequencing. It consists of several cycles where in each one fluorescent-tagged nucleotides with reversible terminators compete to be added in the growing strand. The remaining nucleotides are washed away. Using lasers, the nucleotide is excited, emitting a unique emission frequency that is measured, identifying the corresponding nucleotide. The reversible terminator is cleaved and the cycle starts all over again. Each cluster is sequenced simultaneously, resulting in a massive parallel process, producing each one a read, inferred from the reading of all the signals for every single spot at each cycle. This process is called Sequencing-by-Synthesis (SBS). Now that the forward read is complete, the primer ends are unblocked, the strands are again cleaved but this time the forward strands, already sequenced, are washed away. A new SBS process

begins sequencing now the reverse strands. The resulting data obtained is then analysed.

1.1.2 Roche 454

Small resin beads are deposit in a PicoTiter plate, a small plate containing millions of wells, each one approximately 25 μm in diameter. These beads are covered with adapters, complementary with the ones added to the DNA fragments in the library preparation. The DNA is denatured and the ssDNA binds with the adapters. Due to the nature of the wells, only a single fragment of DNA can be immobilized to each bead. These fragments pass then to a length extension phase, in which each bead will contain a cluster of copies of the same fragment. After amplification, a SBS phase follows. Although, similarly with the Illumina method, sequencing occurs when a nucleotide binds with the template fragment but a different method for recognition is used. This method is based in a enzyme cascade ending with detection of the emitted light. Nucleotide bases, dNTP, are added in waves and if the DNA polymerase adds the nucleotide to the growing chain, a single phosphate, Pi, is released. This phosphate will react with the enzyme luciferase, present in the well. Both of them will react resulting in the products oxyluciferyn and light. This light will be detected by a camera in the machine and saved. Then, the nucleotides are washed and a new cycle starts [12, 14]. This methods allows the addition of multiple nucleotides of the same kind during a single cycle. The technology can detect the multiple additions by the increase of the fluorescent signal intensity, although it can turn into a growth of homonucleotide sequences, resulting in a higher error rate [15, 16]. Apart from this, 454 technology can produce longer reads than Illumina with a high quality. Nevertheless, this type of sequencing is not supported anymore and the reagents are not longer distributed, so it was replaced completely by Illumina sequencing.

1.1.3 Ion Torrent Sequencing

This technology developed by Ion Torrent Systems Inc. and currently owned by TermoFisher, differs from the traditional optic-based sequencers, such as the aforementioned Illumina and Roche 454, on the use of a semiconductor-based approach. Although optic-based sequencers have better accuracy, Ion Torrent sequencing allows lower sequencer costs, less power consumption and higher scalability. During library preparation, adapters are added to DNA fragments that will be later clonally amplified into beads via PCR. DNA fragments are bound to a single Ion Sphere Particle on one side and are biotinylated on the other side. During enrichment process, the

biotinylated side will be bind to magnetic beads that are coated with streptavidin. In this way all empty beads that do not contain any DNA fragment will be washed away so only beads containing template DNA are selected. These beads are pipetted into the semiconductor chip where only a single bead fits inside every well. Remaining beads will be discarded by centrifugation, granting that every well only contains a single bead. Sequencing in this technology proceeds in a similar manner to Roche 454 but instead of using sensors to detect luminescence emitted by enzymes, it detects pH changes in the solution, which means that every well is, in essence, a pH meter. The dNTP are added in a step-wise fashion and if the current nucleotide is complementary to the template DNA fragment, DNA polymerase will add it in the extending strand, releasing H^+ and di-phosphate. This proton will cause a shift in the pH in the solution, proportional to the amount of nucleotides incorporated. The sensor, located in the bottom of each well, will transform this signal from chemical to electrical and digitized. The solution is washed away and the cycle starts again [17]. Such as Roche 454, each cycle can detect homopolymer stretches, leading to higher error rates in this regions [15, 16]. The semiconductor nature also leads to faster and cheaper rapid sequencing speed and low upfront and operating costs, since the detection is made on the chip itself [18].

1.1.4 Different Technologies, Same Output

In the last decades, the development of various technologies and methodologies have led to the advance in sequencing (more data together with a cost reduction). Nowadays, this evolution does not seem to slow down but on the contrary. The price reduction since beginning of the 21st century is well above the Moore's Law prediction, allowing for the use the rapid development of new methodologies for various fields such as evolutionary biology, phylogenomic, population genomic or metagenomic. But progress never stops and the next generation sequencing is already in development. This new technologies, developed by companies like Pacific Biosystems (PacBio) and Oxford Nanopore Technologies, do not require amplification during library preparation, enabling single molecule sequencing with much higher read length [19]. Returning to the current available technologies, despite every technology having their own characteristics, they produce the same information: DNA reads, and the way to express them is in the standard FASTQ format. This is a text-based format for storing biological sequences, such as nucleotides and amino acids. It is a variation of the already existing FASTA format with the inclusion of a PHRED quality score for every single nucleotide. This value is defined in terms of the estimated probability of error.

$$Q_{PHRED} = -10 \times log_{10}(P_e)$$

The quality scores are stored in ASCII, which allows it to be saved in a single character, an efficient storage-saving method, with a high range of probabilities that can be converted to a score between 0 and 126, representing all the characters in a normal keyboard. Since this format is widely extended, most software/pipelines are simultaneously compatible with practically every technology.

1.2 Microsatellites

Microssatellites, or SSRs (simple sequence repeats), are a subcategory of small tandem repeats (1-6 base pairs) consisting of repeated DNA motifs present in eukaryotes and procaryotes that occur in coding and non-coding regions. Their distribution throughout the genome is debated although some studies show the non-random nature of their distribution [20]. What makes SSR interesting is their special nature and their unusual high mutation rate. The mutation rate of this DNA regions varies from specie to specie, but it is estimated to be between 10^{-3} to 10^{-6} per cell generation [21]. Some variables can alter the mutation rates, being difficult to establish a pattern across species. It is known that increased heterozygosity induces higher mutation rates, as well as microsatellite length increase [22, 23].

Due to the nature of microsatellites, they are classified in four categories: perfect, imperfect, interrupted and composite. Perfect pattern consists in a perfect repetitions of the motif (as its own name indicates); an imperfect microsatellite contains a non matching base pair located between the motif; an interrupted pattern contains in the SSR itself a small sequence that does not match the motif sequence; and finally a composite one is formed by 2 different sequences with different motifs next to each other [24].

The majority of SSRs are located in non-coding regions and are relatively rare in protein-coding regions in primates, plants and other clades [25–27]. The relative absence of SSRs in protein-coding proteins can derive form a negative selection against frame shifting mutations [28] and the ones located in the coding region are mostly compose of trinucleotide or hexanucleotide motifs, mitigating the frame shift adjustment.

1.2.1 Mutations Mechanisms

As already mentioned, the mutation rates in SSRs are higher than point mutations in other coding loci which influences its abundance and functions effects associated with the mutations rates. The causes are not well-known but various theories have been proposed to explain SSR instability such as replication slippage and recombination.

1.2.1.1 Replication Slippage

Replication slippage is a replication error that occurs during DNA replication where the template DNA strand and the nascent, that is just growing, suffer from a shift that makes the two strands are different. This phenomena normally happens in highly repetitive areas such as microsatellite regions. This can be caused by the dissociation of one DNA strand from the other, either template or nascent, and rebinding in a different position [20, 29]. As a result, a loop is formed that may cause a contraction or expansion of the SSR, but with a higher bias for expansion [30]. If this error is not detected by DNA repair mechanisms, this event will form a new allele at the SSR loci just copied. The length and purity of the SSR can affect to the mismatch repair mechanisms. Longer and purer have higher mutations rates than shorter and impure SSR [20, 29]. This can be credited to the difficulty of detecting slippage at highly repetitive loci [31]. These events occurs frequently, but if the reparation mechanisms is defective it can result in sequences with tandem repeats becoming prone to mutations, as in tumor tissues, which means that high SSR instability has a pivotal role in carcino-genesis and other genomic diseases such as fragile X syndrome, Friedreich's ataxia and myotonic dystrophy [32, 33].

1.2.1.2 Recombination

DNA recombination was also shown to influence microsatellite mutations. It consists on the exchange of genetic material between chromatids and in eukaryotes this usually takes place during meiosis with chromatids from both parents and in procaryotes in mitosis involving sister chromatids. Although reciprocal exchange of DNA is the most common form, commonly known as cross-over events, non-reciprocal transfer, or gene conversion, has been shown to increase SSR instability, leading to the expansion or contration of SSR [34]. The influence of this mechanism in SSR mutation is under debate, since observations seem to point that this phenomenon does not influence mutability as much as in minisatellites. Minisatellites consist, in the same manner as

microsatellites, of tandem repeats but with repeats of greater length (7 and above base pairs) [35]. In some studies, no significant relation between recombination rate and SSR mutability was found in humans and *Drosophila melanogaster* [36, 37]. However an association between microsatellite occurrence and regions of high recombination rate has been observed in *Saccharomyces cerevisiae* and humans. To conclude, it seems that SSRs distribution is associated with recombination, not as a consequence, but rather because repetitive sequences are involved in recombination [29].

However the regions called *recombination hotspots*, so called because in this regions there are significant higher rates of recombination relatively to the rest of the genome should be taken in consideration. Studies suggest that SSRs can act as recombination hotspots, giving origin to gene conversion events. To back this hypothesis, several SSR were found to be over-represented or strongly associated with recombination hotspots [38–40].

1.2.1.3 Indel Slippage

Indel is the **IN**sersion or **DEL**etion of bases in the genome. It is differentiated from point mutation phenomenons since it does not involve the substitution of a nucleotide. Zhu et al showed that indel-like events tend to duplicate short sequences [41]. Dieringer and Schlotterer proposed a mechanism called indel slippage or length independent slippage, a new class of mutation not encapsulated by standard slippage. Since standard DNA slippage is length dependent, that is, for it to happen, sequences must have a length greater than 8 base pairs [42, 43], indel slippage is by the contrary, length independent, which explains how low repeat sequences emerge [44]. As evidence, it was observed that insertions often copy the flanking sequence, creating a short SSR [29, 41].

1.2.2 Biological Function

To the best of our knowledge, not much is known about the biological function of SSRs. When first described, SSR where classified as non functional DNA due to their unstable nature. The repeat variations were also believed to be neutral with no phenotypic expression, except when associated with neurodegenerative diseases in humans[45]. Nowadays is acknowledged that SSR function is associated with its position on the genome.

When present in the 5' terminal end of a mRNA, also known as 5'- UTR (untranslated region), it may act as protein binding site and improve gene expression when located downstream

from the cap site [46]. In animals, they can also have a significant role in different expression profiles of housekeeping genes (responsible for the maintenance of basic cellular function) and tissue-specific genes [47]. In plants, even with high presence in 5'-UTR regions, their function is not well studied but different SSR types might play different roles in different gene regions [20]. If it is present in the other end of the mRNA,that is, in 3'-UTR they can cause DNA slippage, provoking health disorders and when located in introns, their can take part in the transport and alternative splicing of mRNA, as well as, in conjunction with SSRs located in 5'-UTR, regulating transcription [48].

They are also known for affecting gene expression when located in the promoter or intergenic regions. Their instability can lead to variation in gene expression thanks to changes in transcription factor linkage sites, even to gene silencing. Furthermore, because of their repetitive nature they can alter or modify the secondary structure of the DNA, via the formation of loops or altering the chromatin structure [49].

In a more wide prespective they can have an effect in species adaptation and evolution. Being one of the most mutable site in the genome they can quickly generate novel variation to aid specie survival and adaptation to hostile environments. In fast growing organisms, such as bacteria, the maintenance of numerous SSRs can lead to relative high variability, helping with the rapid response to environmental changes and compensation for the loss of genetic diversity due to genetic drift and selection. We can infer that such events can also occur in other organisms, such as plants and animals, but to a lesser extent [49].

1.2.3 Microsatellite Genesis and Death

The generation of new SSRs is still controversial although the consensus is that, at least, two not mutually exclusive phenomenons can originate new SSRs: *de novo* SSR, and adopted SSRs.

de novo SSRs can be created when a sequence lacking a clear tandem arrangement suffers a mutation, creating a proto-SSR, a short sequence containing 2 to 4 repetitions of the same motif. These mutations can occur as a result of a base substitution and indel-like events, such as the previously mentioned indel slippage [41, 50].

Adopted SSRs, on the other hand, take advantage of mobile elements present in the genome. These are a type of genetic material capable of moving through the genome, that is, pieces of sequence able to being integrated in a new site within the cell of origin, wich are very common in

eukaryotic cells [51]. It is believed that these sites contain one or more sites predisposed to the formation of SSR, thus favoring the dispersion of SSRs through out the genome [29]. SSRs were shown to be associated with mammal SINE and LINE elements, a subclass of mobile elements. These elements have adenin-rich regions sensitive to reverse transcription errors (process where RNA from retrotransposons, such as SINE and LINE, are converted into DNA to be integrated back to the genome), which in turn can lead to the genesis and expansion of A-rich proto-SSRs, should slippage occur [50]. Other proposed mechanisms were also the extension of the 3' end of retrotranscripts, instead of mutation of retrotransposons, in a similar process to mRNA polyadenylation [52].

During its "life" contractions and extensions affect the SSR but, when in equilibrium, the SSR is maintained although a combination of two processes can lead to the "death" of the SSR. If a mutation interrupts the motif repeat it can lead to SSR stabilization by inhibiting the formation of mutational intermediates, such as loops, blocking slippage events and/or by altering intermediates so it can be more easily detected and repaired [53]. Successive interruptions can significantly alter the SSR turning it in a blend of unique DNA sequences that includes only short segments of the original repeat array. Large deletions can also occur on this interruptions what would lead to the loss of large sections of the SSRs, accelerating the process of degeneration. The death of a SSR is a slow and multi-phasic process that can last multiple generations and needs a very drastic set of circumstances to happen. Because of this, is believed that the death rate is much lower than the birth rate, which explains the enrichment of SSRs in eukaryotic organims [29, 50]. The final result is a collection of random and scrambled DNA, very similar to the sequences that are believed to originate SSRs in the first place. This observation drove to the creation of the SSR life cycle hypothesis in which a dead SSR locus could potentially resuscitate [50].

1.2.4 Applications

The high levels of polymorphism in SSRs, in junction with its multi-allelic nature, codominant inheritance, small length, extensive genome coverage, relative abundance and requirement for only a small amount of starting DNA, allowed its successfully application to a wide variety of research fields and practical disciplines [54].

1.2.4.1 Gene Mapping

One of the major fields in which SSRs are heavily used is genetic mapping. Genetic mapping consists in the localization of genes underlying phenotypes on the basis of correlation with DNA variation, without the need for prior hypotheses as opposite to the candidate gene approach [55]. In eukaryotes, the process of meiosis will split each pair of chromosomes between the two sister cells, the two gametes. Genes that are not located in the same chromosome are inherited independently, where genes located on the same chromosome are inherited together. Even so, if cross-over occurs, it can result in the physical exchange of genetic material between two homologous chromosomes. As such, genes that are located closely to each other have high probability of being inherited together, that is, the more closely a gene or marker is located to other, the higher the "linkage" between them is. When SSRs are located in gene coding regions (and as such "linked" to the gene), they directly show the location of the associated gene within the linkage map, often representing genetic variations associated with significant phenotypes. They can also be very useful in marker assisted selection (MAS) in commercial important species [56, 57]. If they are present in non-coding regions, they are very helpful in building a dense linkage map framework into which coding markers are incorporated [56, 58].

1.2.4.2 Individual Identification

SSRs represent codominant single-locus markers. The SSRs present in an individual genome are a combination of two alleles, one from the male parent and the other from the female parent. As such, and using a panel of several SSR loci, a unique combined SSR genotype profile can be produced for each individual tested, perfect for paternity and relatedness analysis of natural populations [58]. They can also be used in forensic science, since its small size makes them relatively more resilient in degraded DNA. For example, SSRs remain relatively stable in bone remnants and dental tissue, providing the basis for the successful application of ancient DNA for molecular analysis. Its utility can also be extended to illegal poaching, food fraud and other economical crimes [58–61].

1.2.4.3 Phylogeny

SSRs can be also used to infer phylogenetic relationships at levels bellow species level or recently diverged species since its variation evolves significantly more quickly that other genomic

regions. However, homoplasy can occur, undermining the confidence of the inferred phylogenetic hypotheses [58]. Homoplasy is a shared character, or in this case SSR, that did not arise from a common ancestor due to convergence or parallelisms. In other words, homoplasy is a false homology [62]. The primers used for SSR amplification can also be problematic since primers from one taxon my not work in other *taxa*. As an alternative, the flanking regions are useful to establish phylogenetic relationships between species and families since they evolve at a much slower rate than the number of repeats in a SSR [58, 63]. For studies of different populations of the same species in short geographical distances, SSRs are ideal, being used to study panmixia, fragmentation, hybridisation, population structure and others [58, 64–66].

1.2.4.4 Conservation efforts

Endangered species will have small and/or declining populations, causing the inevitable loss of genetic diversity and inbreeding. Inbreeding reduces reproduction and survival rates and loss of genetic diversity, reducing the ability of populations to endure environment changes and to evolve and contributing to extinction risk, specially in small populations of threatened species [67]. Conservation strategies focus on using biparental polymorphism of nuclear DNA to reflect characteristics to cope with environment conditions. In this manner, SSRs can be utilized to avoid crossbreed between very close individuals, avoiding the production of weaker generations and the accumulations of older and harmful alleles and to encourage the breeding of different populations to improve species fitness and survival in particular habitats.

Chapter 2

State of the Art

2.1 SSR_Pipeline

SSR_Pipeline [68] is a python coded Pipeline design to identify simple SSRs presents in FASTQ formatted files. Given the high volume of data produced by illumina platforms, compressed files, such as *gzip*, can also be used as input. The workflow of this pipeline allows filtering of subsets of pair-end sequences that do not pass illumina quality control. Next, using a python coded implementation of FLASH, both pair-end sequences, R1 and R2, are merged together into a single DNA sequence. After, using a custom script, a SSR search is made. The user can stipulate the motif size, minimum repetitions and flanking regions size to be used by the SSR search algorithm. If only this script is used, it can also accept FASTA formatted files as input. Since the number of SSRs detected by this pipeline can be huge, no primer design options are performed.

2.2 GMATA

Genome-wide Microsatellite Analyzing Tool Package (GMATA) [69] is a package created for SSR mining, statistics analysis and plotting with added options as well for marker analysis. The long DNA sequences to be used, in FASTA format, are chunk in a more appropriate size to increase mining speeds. Using Perl regular expression patterns, basic SSR motif library will be created. This library will be later used for SSR greedly research, using Perl pattern matching algorithm. At this point the user can adjust the parameters to be used, such as motif size and number

of motif repetitions. An option for highlighting microssatelites in the target DNA sequence is also available. After the search, a statistical classification and summarization is computed. With all this information, the primers are designed by extrating the flanking segments of the detected SSRs and parsing them to Primer3 [70]. The resulting primers are saved in NCBI's STS format. Finally a e-PCR is executed to verify if any primer polymorphism exists. This pipeline is available for most of the popular operative systems such as MAC OS, Linux and Windows 7, 8 and 10. A GUI is also available but it can also be executed through command-line.

2.3 Full_SSR

Full_SSR [71] is a simple pipeline created for SSR search and Primer design. It was programmed in Perl and consists of two modules. The first one, *SSR_Search*, creates a library of all combinations of 2 to 5 nucleotides. Then, it searches for all the combinations present in the motif library in the fasta file input. The output from this module returns the sequence ID as well as the length, start position, end position, motif, number of motif repetitions of the detected SSR. The second module *Primer_Design*, uses a modification of BioPerl package, parsing the significant results to Primer3. No further processing is made. This simplified pipeline is only compatible with Linux operating systems.

2.4 SSR-Primer Generator

SSR-Primer Generator (SSRPG) [72] is a web application for identifying SSRs, design primers and for homology-based search of *in silico* amplicons from a DNA sequence dataset. Using a JRE (Java runtime Environment) it analyze Multi-FASTA files in search of SSRs using SPUTNIK [73]. It returns a summary file of the found SSRs that includes the repeat types identified, the motifs involved, positions, length, repeat scores, and flanking regions. Then, this information is parsed to Primer3 [70] for primer design. In this pipeline, Primer3 settings file is set by default and cannot be modified by the user. The resulting primers are filtered to increase their stability by discarding primers pairs with simple repetitive DNA sequences and cross-homologous primer pairs showing high sequence similarity. Finally, primers are parsed into BLASTX to discover if the resulting amplicons are related to protein-coding regions. The output is presented in a HTML file containing the SSRs, SSRs-primers and BLAST results.

2.5 SSR Locator

This Perl/Delphi codded utility [74] integrates SSR searches, occurrence frequency for motifs and arrangements, primer design, and PCR simulation against other databases. Firstly, a SSR search is performed in FASTA formatted files using a similar algorithm to MISA [75] and SSRIT [76] with minor changes. A Delphi script eliminates overlaps of 2 adjacent SSRs. The resulting output is parsed then to Primer3 for primer design and after a Virtual-PCR the annealing sites are found for the two primers. Flanking regions and primer sequences are moved to a new variable that will be aligned with the original amplicon to check the correspondence in the sequences predicted.

2.6 Limitations

Although each program tries to facilitate and automatize SSR search and primer design, there are several limitations on the reviewed and available software. These limitations result in part due to the different purposes for which the software was created for. As an example, Full_SSR, SSR-Primer Generator and SSR Locator were created to use data already published rather than raw data. Next follows a more detailed description of the most important limitations detected.

First difference between both programs is found in the input file type. Only SSR_Pipeline accepts files obtained directly from next generations sequencing (NGS) technologies as input. On the contrary, some protocols only accept FASTA files, what means that data obtained from NGS technologies need to be pre-processed, which results in time wasting and manual interference.

SSR_pipeline has a very simplistic approach to SSR search and the output resulting from the analysis is hard to read/analyze without the use of a custom script. Also the lack of further processing and primer design limits the usefulness of this tool. Full_SSR and SSR Locator also have the same limitations as SSR_pipeline, having a very simplistic nature without any kind of filtering, not being suitable for the task at hand.

In SSR-Primer Generator the user is restricted from altering primer3 settings, that is, it cannot be changed, preventing the tool from shaping to user needs. The later stages of this software also illustrate its focus on associating SSRs to gene expression.

GMATA is most similar software in purpose to Micro-Primers but it contains a great amount

of output files that, for the current objective, does not provide any useful information, moreover, it increases run time and computational power. It also has a major flaw, also present in the remaining software.

There is the possibility that different alleles found are, in fact, the same SSR. An allele is every single variation of DNA in the same genetic regions. No reviewed software performs, for example, an allele clusterization to truly identify each individual SSR exiting in the population sample.

Although some of these programs could be adapted to create primers for the found SSR, still it should be necessary to include extra manual steps to reduce the volume of data to single SSR loci since none of them take into account the multiple alleles every SSR has. This process would result in a extra time and resources consumption and would fail on the purpose of being an automatic pipeline. The set of limitations above mentioned were the main motivation for the creation of Micro-Primers.

Chapter 3

Motivation and Methods

Nowadays, it can take up to months, for laboratories and researches to analyze and select SSR primers for their projects. It requires them to run several times a primer detection software without any kind of pre-processing and to select by hand the individual candidates. Without clustering or knowing the number of available loci, most amplifications are inefficient, time consuming and more resources are wasted on reagents. The goal of this work is to design and implement a novel tool, named Micro-Primers, that automatically executes the screening of SSRs in a set of raw sequences generated by RAD-seq (technology where sequence fragments are generated by restriction enzyme that cuts always in a particular pattern, leaving in both ends a known sequence [77]) and to design primers for SSR amplification. The necessary software to search and create primers for the SSRs already exists and are widely used in the scientific community. However, as those programs are meant to be used individually, each program has its own specific input and output. For example, to combine the output of a program A with the input of another program B, researches must format manually the output of A so it can suit the input of B. Thus, this restriction forces researchers to continuously be aware of the execution progress, to format each output manually and to select possible candidates from a database of thousands of unique sequences. Micro-Primers tool provides a framework, where those programs are aggregated sequentially in a single automated pipeline, that deals with the low-level communication details between them.

3.1 Micro-Primers composition

Micro-Primers pipeline was written in Python version 3.6 and consists basically in two different scripts: (i) `install.py` that includes all necessary pre-requisites for a proper installation of Micro-Primers, and (ii) `micro-primers.py`, the pipeline itself. Analysis settings are described in the `config.txt` file, where parameters can be modified by the user accordingly to his own particular needs. The folder `software`, provided together with the Python scripts, contains all the scripts and external software employed by `micro-primers.py` with all the external programs needed for a correct execution.

3.2 Input Files

In order to run Micro-Primers, users only need to provide two FASTQ files corresponding to both ends of a pair-end sequencing. Samples should come from a pool of (untagged) individuals of the same species so the microsatellite selection can be optimized. SSR selection will be performed based on the number of alleles of each SSR loci, so the more heterogeneous is the sample, the better will be the final result. Reads must come from a microsatellite library built using a restriction enzyme and following an enrichment protocol such as the one described in [78]. The idea behind the enrichment protocol after digestion is to have a random representation from the whole genome where the target SSR motifs will be the most represented strands in the final library. A fragment size selection is then performed on the enriched library to keep only fragments of an average length lower than the maximum sequencing length so both paired-ends reads overlap when merged later on. The final fragment size is important for microsatellite screening and must comprise the full SSR pattern (variable in length) and the two flanking regions with fair length for primer design.

3.3 Alleles

There is the possibility that different alleles found are in fact the same SSR. An allele is every single variation of DNA in the same genetic region. In microsatellites, they correspond to different number of repeats of the same pattern in the tandem sequence. Because of this, it is necessary for each SSR found to be clustered, each representing the different alleles found in the sample for

a given SSR. The number of alleles found for each SSR is meaningful since it may indicate that the specific SSR is well-established in the population. Thus, the higher the number of alleles is for a SSR, the less number of SSRs is needed to characterize a population or individual.

Normally an allele with low frequency in the sample is discarded. If, during a normal analysis, the number of qualifying alleles is low, it is possible to re-run the analyses and obtain meaningful alleles. The simultaneous presence of few alleles with a large variation of motif repetitions can suggest the presence, in the natural populations, of individuals with alleles with motif repetitions between those found. Micro-Primers allows the execution of a special search where, if activated, the difference between the alleles with lowest and highest repetitions for each SSR is calculated, thus ignoring the total number of alleles found. Those whose difference is above or equal to the user defined minimum, move along the pipeline normally. It's recommended to only use this option after an unsuccessful or below expectations run.

3.4 Workflow

In the initial stage, all input files are pre-processed (Figure: 3.1–I). Firstly Trimmomatic [79] is used to remove technology specific adapters, left-overs from the sequencing. Then, Cutadapt [80] follows which again removes left-over adapters, this time belonging to restriction enzymes adapters, remainders of library preparation. These programs, Trimmomatic and CutAdapt, already use fastq format and, as such, there is no need for any kind file processing. As a result of the pair-end sequencing technology, the reads obtained are divided into forward reads and reverse reads, which allow a more effortless and precise alignment [81]. Since the protocol followed does not require any kind of sequence alignment, these two complementary reads must be combined into a single sequence. For this, FLASH [82], a fast-computational tool, is used to merge the pair-end reads into a single sequence. With this concludes the pre-processing phase.

Next, all sequences are filtered. Both ends of each sequence are analysed in search for the enzyme restriction pattern. Only sequences containing the restriction enzyme pattern will be exported to a new FASTA formatted file (Figure 3.1–II). FASTA is a text-based format for representing either nucleotide or protein sequences. This format starts with a unique line, starting with the symbol greater-than (">"), with a sequence descriptor (ID) which contains a summary description of the sequence. In this pipeline, since the sequences derive from FASTQ format files, the sequence identifier will be identical to those present in the original input file. The following

lines represent the sequence, which represent the same standard IUB/IUPAC nucleotide code as the FASTQ. The original sequencer IDs are big and complex and, as such, new numerical ID must is added to each sequence for easier identification. This is done by a custom script, *add id calculate length* (Listing 4.19). It will add a unique integer followed by a underscore "_ " to each sequence ID. The addition of an underscore will allow to easily split the ID from the original identifier, while saving the sequence description which can be useful for the user. This script will also calculate additional parameters, necessary for later filters (Figure 3.1–III). In this case, sequence length is computed.

Afterwards, sequences are ready to be parsed to MISA [75]. It is a Perl script used for identification and location of microsatellites in nucleotide sequences. MISA will output a matrix containing for each sequence (i) ID + sequence description; (II) SSR found; (III) SSR type; (IV) SSR Motif and number of motif repetitions; (V) SSR start position; (VI) SSR end position.

Following the SSR search, parameters previously computed are attached to the SSR matrix using *length merger* (Listing 4.20). This additional data will be used to evaluate if regions surrounding the SSR, also called flanking regions, are long enough to accommodate SSRs primers for SSR amplification. This constitutes the second filter in Micro-Primers (Figure 3.1–V). It is important that filters along the pipeline are added since not every SSR found can be a proper candidate for future amplification. For example, SSRs with complex patterns or of large size cannot be of interest to the researcher and their presence can significantly slow the pipeline. Each filter should have a range of values and thresholds that can be altered, giving the user total control to select which SSR best suits his needs.

Since multiple alleles can be in fact the same SSR (Section 3.3), further processing is needed. Each allele found should be clustered so each individual SSR can be identified as unique. It is known that different SSRs can have the same motif. To avoid false positives, only the flanking regions are used for clustering. Thus, *split* (Listing 4.21), is used to extract the SSRs from the original sequences (Figure 3.1–VI). Subsequently, a clustering with all the flanking regions is performed by CD-HIT [83, 84] (Figure 3.1–VIII). CD-HIT can handle huge datasets with millions of sequences and can be hundreds of times faster than other more common methods, such as BLAST. The result of this process will be a reduction of the redundancy. Every single sequence will be assigned to a cluster and the number of elements of every cluster is calculated. This two new features will be added to the SSR matrix using *add cluster info*.

After clustering, non-polymorphic SSRs might be present in the sample being useless for

individuals or population identification. For that reason all SSR clusters move through the third and final filter (Figure 3.1–IX) where they are evaluated to verify if the number of alleles is above user specifications (Listing 4.15).

Given the nature of SSRs, only a single representative from each cluster is needed for primer design since the region of interest for primer design is the flanking region which is common to every allele in the cluster. Under this premise, a random allele is chosen from every cluster as representative (Figure 3.1–X). If no suitable candidates are found, a looser search will be executed. This search will take in account the difference between the shortest and longest allele in each cluster and if the difference is big enough, it is expected that, in the wild, individuals may have also alleles within this size range, increasing the candidate pool (Section 3.3).

After the third filter, all the requirements for primer design are met. All the information available for each selected SSR is aggregated into a file properly formatted to subsequently be parsed into Primer3 [70] (Listing 4.25). Primer3 is suitable for being used in high-throughput pipelines for genome-scale research, what makes it a great tool for this pipeline (Figure 3.1–XI). Despite its utility, the resulting Primer3 output is not user friendly, meaning that the output should be altered to be presented in a straightforward, concise and simple format, ready to use (Figure 3.1–XII). A custom script, *final_primers* (Listing 4.26), perform this conversion, resulting on the final result of the pipeline. It contains all the necessary information for SSR amplification such as (i) sequence ID; (ii) primer product size; (iii) forward primer; (iv) forward primer Tm; (v) reverse primer; (vi) reverse primer Tm; (vii) SSR motif; (viii) and number of alleles found. The flag "|BEST|" can also be shown at the end of each line, advising the user which SSR primer pair should have better performance.

3.5 Execution parameters

All the parameters that Micro-Primers needs to properly perform the analysis must be set at the `config.txt` file. In this text file there are four sections with different parameters to take into account for the pipeline execution.

First of all, it can be found the "Input files" section where the user has to indicate the name of the pair-end files that will be used in the analysis. Then, at the "CUTADAPT" section, the sequence of adapters used after the restriction enzyme digestion is required.

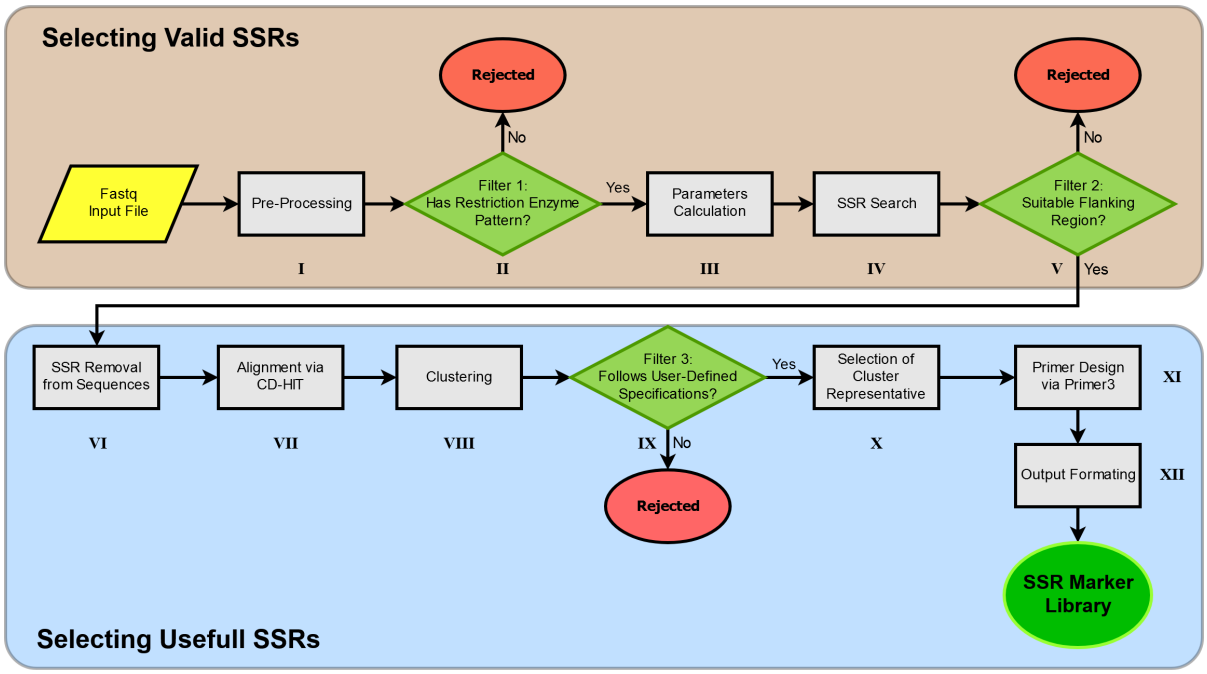

Figure 3.1: Flowchart of Micro-Primers.

These adapters were necessary to transform the longer overhangs into blunt ends after the enzyme digestion. Next stage is called "SSR" and several parameters regarding the microsatellite selection are involved. The parameter MIN_FLANK_LEN indicates the minimum length accepted in both flanking regions where the primers will be designed on.

PCR amplification primers usually have a length around 20-25 nucleotides and some particularities are required, like the presence of G or C at the 3' end, certain percentage of GC for a proper melting temperature, both primers should have similar melting temperature for the hybridization to take place at the same time [85]. All these factors make critical the length of the flanking areas since a very narrow window can make impossible the primer design and subsequently the SSR to be discarded. Thus, every sequence with shorter flanking region in one or both ends will be discarded.

The MIN_MOTIF_REP sets the minimum number of repeats that every SSR region must have to continue in the pipeline. In the EXC_MOTIF_TYPE parameter, the user can discard some SSR motifs for not being analysed. Options for this parameter are c;c*;p1;p2;p3;p4;p5;p6, being compound, compound with imperfection, or motif with different repeats from 1 to 6 nucleotides, respectively. It should be noted that bigger motifs are supported but any higher than 6 bases does not classifies as an microsatellite, but as a minisatellite.

Motifs selected previously that must be discarded should be indicated separately with

comma. At the MIN_ALLEL_CNT option the minimum number of alleles for a SSR loci to be selected is indicated. The parameter under the name MIN_ALLEL_SPECIAL is used to enable or disable (1=enabled, 0=disabled) the option to select microsatellites with less number of alleles previously indicated as long as the difference between the allele with higher number of repeats and the allele with a smaller number of repeats satisfy the number indicated in the MIN_ALLEL_SPECIAL_DIFF. Last section in the `config.txt` file is about PRIMER3. Here the only requirement is to indicate the path to the Primer3 settings file. In this settings file several parameters can be changed but a general one is provided with the standard parameters that Primer3 includes. Table 3.1 shows the parameters that can be configured by the users. As one is able to observe, no spaces or additional text character should be present after the character "=". Every setting, besides R1 and R2 file names, have a default value defined in the configuration file.

Table 3.1: Settings to be configured by the user.

Parameter	Input accepted	Description
INPUT_FILES_R1	file directory	File name and directory of R1 fastq file to be analysed. Final output will be named after this file.
INPUT_FILES_R2	file directory	File name and directory of R2 fastq file to be analysed.
CUTADAPT_3	string	Reverse Adapter to be cut from sequence by CutAdapt.
CUTADAPT_5	string	Froward Adapter to be cut from sequence by CutAdapt.
MIN_FLANK_LEN	integer	Minimal length of SSR flanking regions.
MIN_MOTIF_REP	integer	Minimal number of SSR motif repetitions
EXC_MOTIF_TYPE	c;c*;p1;p2;p3;p4;p5;p6	Motif type to be excluded. | c,c* - compost; px - x bases motif
MIN_ALLEL_CNT	integer	Minimal number of individual alleles
MIN_ALLEL_SPECIAL	0;1	Special search activation. | 0 - Deactivated; 1 - Active
MIN_ALLEL_SPECIAL_DIF	integer	Minimal difference between highest and lowest allele repetition among found SSRs .
PRIMER3_SETTINGS	file directory	File name and directory to Primer3 settings file.

Chapter 4

Implementation Details

In the previous chapters, we motivated the need for automatic management of input and output files from the different software aggregated, and explained that each one has unique and different specifications. This lead to the creation of this pipeline and the supporting functions.

In this chapter, we show the internals of the Micro-Primers software. To do so, we show its procedures, a description of their role in the pipeline and a detailed explanation of its inner workings.

4.1 Pipeline Control

4.1.1 Setup

```
1  import os, sys
2  from software.scripts import picker, config, text_manip, pre_primer
3  #System call handler
4  def micro_primers_system_call(cmd, erro):
5      rvalue = os.system(cmd) # returns the exit code
6      if rvalue != 0:
7          sys.exit(erro)
8  #Defining output name
9  def name(file_name):
10     output_name = file_name.split("_R") + "_primers.txt"
11     return output_name
12 #Check if primer3 input is empty
13 def size_check(SPECIAL_CASE):
14     if os.path.getsize(".temp/primer3_input_out.fasta") < 1:
15         print("Empty primer3 input file. \n")
16         if SPECIAL_CASE == 0:
17             sys.exit("No valid SSRs were selected. Try to use a broader
                   search.")
18         sys.exit("No valid SSRs were selected.")
19 #Removal of .temp directory
20 def junk():
21     micro_primers_system_call("rm -r .temp/", "Error: Could not remove .temp
           directory.")
22 #Create empty file for importing python scripts
23 micro_primers_system_call("touch software/scripts/__init__.py", "Error: Could
       not create init.py.")
24 #Reading settings
25 settings = config.config("config.txt")
26 #Creation of hidden temp file
27 def folder (folders):
28     for i in folders:
29         if os.path.isdir(i) == False:
30             micro_primers_system_call("mkdir %s" %(i), "Error: could not
                   create %s directory." %(i))
```

Listing 4.1: Auxiliary procedures to support pipeline stages

The script responsible for pipeline control is divided in 3 parts: (i) setup; (ii) Function Definition and (iii) Function Call.

The setup phase is very straight-froward. At the beginning, all functions located in *scripts* folder are imported (line 2). The function *micro primers system call* is also defined. It will process any host operating system call and will handle any errors. If an error occurs during a Micro-Primers call, it will display a short message with the error description (lines 7). The function *name* will define the output file name from the R1 input file name, which will contain the designed primers (lines 8–11).

It is possible, due to an error or non-selection of valid SSRs, that the input file created for Primer3 might be empty and parsed into it, without any error or pipeline stop. To avoid pipeline crash, *size check* will check file size. If an empty file is detected the user will be advised to either activate "special search" or loosen SSR specification. Next, function *junk* is defined, which will erase any intermediate files (lines 13–21).

Finally, the *config* file is loaded, in which each setting is loaded as a variable into the script, with the help of the function *config* previously imported (line 25). The folders *.temp* and *logs* are also created if not found in the main pipeline directory. They will save respectively the temporary files and log files created during Micro-Primers run (lines 27–30).

4.1.2 Function definition

Afterwards, the needed functions for the correct operation of Micro-Primers are defined.

```
#Sequences Triming of Sequencer adapters
def trimmomatic(R1, R2):
    print('Trimmomatic working...')
    micro_primers_system_call(
        "java -jar software/Trimmomatic-0.36/trimmomatic-0.36.jar PE -phred33 "
        "%s %s "
        ".temp/trim_out_trimmed_R1.fastq .temp/trim_out_unpaired_R1.fastq "
        ".temp/trim_out_trimmed_R2.fastq .temp/trim_out_unpaired_R2.fastq "
        "ILLUMINACLIP:./software/Trimmomatic-0.36/adapters/TruSeq2-PE.fa:2:30:10"
        " LEADING:3 TRAILING:3 SLIDINGWINDOW:4:15 2> logs/trim_log.txt"
        %(R1, R2),
        "Error: Trimmomatic could not remove adapters")
#Adapters removal (specifics from technology)
def cutadapt(a, g):
    print('Cutadapt working...')
    micro_primers_system_call("cutadapt "
                "-a %s "
                "-g %s "
                "-o .temp/cut_out_nolink_R1.fastq
                    .temp/trim_out_trimmed_R1.fastq "
                "> logs/cut_log_r1.txt"
                %(a, g),
                "Error: CutAdapt could not remove adapters from R1 sequence.")
    micro_primers_system_call("cutadapt "
                "-a %s "
                "-g %s "
                "-o .temp/cut_out_nolink_R2.fastq
                    .temp/trim_out_trimmed_R2.fastq "
                "> logs/cut_log_r2.txt"
                %(a, g),
                "Error: CutAdapt could not remove adapters from R2 sequence.")
```

Listing 4.2: Procedures for Trimmomatic and CutAdapt stages

```
# Fusion of R1 and R2 files
def flash():
    print('Flash working...')
    micro_primers_system_call("software/FLASH-1.2.11/flash "
              ".temp/cut_out_nolink_R1.fastq "
              ".temp/cut_out_nolink_R2.fastq "
              "-M 220 -o .temp/flash_out 2>&1 |"
              " tee logs/flash.log "
              "> logs/flash_log.txt",
              "Error: FLASH could not merge sequences.")

# Selecion of fragments that start and ends with the pattern of the
    restriction enzyme
def python_grep():
    print('Selecting sequences with restriction enzyme patterns...')
    text_manip.python_grep(".temp/flash_out.extendedFrags.fastq",
        ".temp/grep_out.fasta")
```

Listing 4.3: Procedure for FLASH and python grep call

```
#Change ids and Length Calculation for later selection of valid microsatellites
def ids_and_len():
    print('Adding ids...\nCalculating sequences lengths...')
    text_manip.add_id_calculate_length(".temp/grep_out.fasta",
        ".temp/ids_out.fasta", ".temp/length_calc_out.fasta")
```

Listing 4.4: Procedure to attribute a unique ID and calculate total sequence size for each sequence.

```
#Search Microsatellites
def misa():
    print('Misa working...')
    micro_primers_system_call("perl software/scripts/misa.pl "
              ".temp/ids_out.fasta "
              "2> logs/misa_log.txt",
              "Error: Misa failed")
```

Listing 4.5: Procedure for the MISA call (SSR search).

```
1 #Adds length to end of the sequences to misa output
2 def length_merger():
3     print('Adding length to MISA output...')
4     text_manip.length_merger(".temp/misa_out.misa",
          ".temp/length_calc_out.fasta", ".temp/length_add_out.misa")
```

Listing 4.6: Procedure adding length to MISA output.

```
1 #Selection of microsatellites with enough space for primer
2 def good_micros(MIN_FLANK_LEN, MIN_MOTIF_REP, EXC_MOTIF_TYPE):
3     print('Selecting good microsatellites...')
4     picker.csv_picker(".temp/length_add_out.misa",
          ".temp/good_micros_out.fasta",
5                 ".temp/good_micros_table_out.misa", MIN_FLANK_LEN,
                      MIN_MOTIF_REP, EXC_MOTIF_TYPE)
```

Listing 4.7: Procedure for the *csv picker*.

```
1 #Extraction of the microsatellite sequence from alignment of fragments with
      flanking regions
2 def splitSSR():
3     print('splitSSR working...')
4     text_manip.split(".temp/good_micros_out.fasta", ".temp/ids_out.fasta",
          ".temp/split_out.fasta")
```

Listing 4.8: Procedure for splitting SSR.

```
#Removal of Redundacy
def cdhit():
    print('CD-HIT working...')
    micro_primers_system_call("software/cdhit/cd-hit-est "
              "-o .temp/cdhit_out.txt "
              "-i .temp/split_out.fasta "
              "-c 0.90 "
              "-n 10 "
              "-T 10 "
              "> logs/cdhit_log.txt",
              "Error: CD-HIT failed")
#Cluster assignement
def cluster():
    print('Calculating number of sequences for each cluster...')
    text_manip.cluster(".temp/cdhit_out.txt.clstr", ".temp/clusters_out.txt")
#Adding cluster information to Microsatellites table
def cluster_info():
    print('Adding information to the table of microsatellites...')
    text_manip.add_cluster_info(".temp/clusters_out.txt",
        ".temp/good_micros_table_out.misa", ".temp/cluster_info_out.txt")
```

Listing 4.9: Procedure for allele clusterization.

```
def cluster_filter(MIN_ALLELE_CNT, MIN_ALLELE_SPECIAL, MIN_ALLELE_SPECIAL_DIF):
    picker.allele(".temp/cluster_info_out.txt",
        ".temp/cluster_filter_out.txt", MIN_ALLELE_CNT, MIN_ALLELE_SPECIAL,
        MIN_ALLELE_SPECIAL_DIF)
# Selecting one sequence per cluster
def selected_micros():
    print('Selecting one sequence per cluster...')
    picker.selected_micros(".temp/cluster_filter_out.txt",
        ".temp/selected_micros_out_seqs.txt",
        ".temp/selected_micros_out_tabs.txt")
```

Listing 4.10: Procedure for cluster filtering and choice of loci representative.

```
#Creating input file for Primer3
def primer3_input():
    print('Creating Primer3 input file...')
    pre_primer.pseudofasta(".temp/selected_micros_out_seqs.txt",
        ".temp/ids_out.fasta", ".temp/primer3_input_out.fasta")
#Primer design and creation
def primer3(p3_settings):
    print('Creating Primers...')
    micro_primers_system_call("software/primer3/src/./primer3_core "
                "-default_version=2 -p3_settings_file=%s "
                ".temp/primer3_input_out.fasta "
                "-output=.temp/primer3_out.primers "
                %(p3_settings),
                "Error: Primer3 failed")
#Selection of primers following laboratory criteria and output formatting
def output():
    print('Selecting best primers...')
    print('Creating final file...')
    pre_primer.final_primers(".temp/selected_micros_out_tabs.txt",
        ".temp/primer3_out.primers", name(settings[0]))
```

Listing 4.11: Procedure for primer design and output formatting.

Every function follows the same template (Listing 4.2 to Listing 4.11). First, a comment describes function purpose. Second, a *print* call is made to inform the user, in real time, about current pipeline progress. Third, the functions are defined with their arguments. All outputs also follow a common template. Every output is named by the combination of the function name responsible for its creation and the prefix "*_out*". This improves readability, overall organization, and error detection since a incorrectly formatted or blank file can be pin-pointed to the function at fault.

In case of third-party software calls, such as Trimmomatic and Cutadapt (Listing 4.2), MISA (listing 4.5), CD-HIT (listing 4.9) and Primer3 (listing 4.11), the necessary additional arguments and error messages are added.

4.1.3 Function call

Finally, in the third step, all functions are called.

```
#Pipeline
folders([".temp/","logs"])
trimmomatic(settings[0], settings[1]),
cutadapt(settings[2], settings[3]),
flash()
python_grep()
ids_and_len()
misa()
length_add()
good_micros(int(settings[4]), int(settings[5]), settings[6])
splitSSR()
cdhit()
cluster()
cluster_info()
cluster_filter(int(settings[7]), int(settings[8]), int(settings[9]))
selected_micros()
primer3_input()
size_check(int(settings[8]))
primer3(settings[10])
output()
junk()
print('Done!')
```

Listing 4.12: Pipeline with function calls and their corresponding settings.

Each individual function can be silenced with the addition of the comment symbol "#" at the start of the line. This gives the pipeline a modular nature, allowing the user to control which function will be called. As an example, the user can choose to pre-process the sequences using alternative software to Trimmomatic and CutAdapt and run the rest of the pipeline normally (although this option involves extra steps, something that will be improved in next implementations). If any error occurs, the function *junk* can also be silenced to allow intermediary file analysis for error detection.

Finally a "Done!" message is shown, signalling the end of the pipeline.

4.2 Filters

4.2.1 matrix_picker

```
def matrix_picker(rf,
                  of_micros_good,
                  of_micros_tab,
                  min_ext_dist,
                  min_rep,
                  exclude_ssr):
    readfile1 = open(rf, "r")
    outfile1 = open(of_micros_good, "w")
    outfile2 = open(of_micros_tab, "w")
    ID = 1
    for line in readfile1:
        selected_line = line.split("\t")
        #Select line which do not contain the correct SSR motif type.
        if not selected_line[2] in exclude_ssr:
            #Remove second "_" from ID. It messes with splitSSR script.
            remove_under = list(selected_line[0])
            for i in range (10, (len(remove_under))-1):
                if remove_under[i] == "_":
                    remove_under[i] = " "
            selected_line[0] = "".join(remove_under)
            # Selecting only sequences that have at least 50 bases before and
                after the SSR
            if int(selected_line[7]) - int(selected_line[6]) >= min_ext_dist
                and int(selected_line[5]) >= min_ext_dist:
                good_micros = list(selected_line[i] for i in [0, 5, 6, 7])
                good_micros = [str(ID)] + good_micros
                outfile1.write("\t".join(good_micros))
                outfile2.write("\t".join(selected_line))
                ID += 1
    #Adding "\n" to the end of the file. It alsos messes with splitSSR script
    outfile1.write("\n")
    readfile1.close()
    outfile1.close()
    outfile2.close()
```

Listing 4.13: Procedure to select sequences that follow the user specification.

Listing 4.13 refers to the second filter present in the pipeline. This function is responsible for filtering the sequences that obey to user specifications (Figure 3.1 – II). In the first lines the input and output files are open (lines 7–9). Next, all sequences are analysed and only that ones that do **not** contain the SSR motif will be excluded (line 15). After minor string corrections (lines 17–21), the size of the flanking regions is calculated. If the flanking region length is over user specifications, a new identifier is added. If not, the sequence is discarded. This second identifier is added for the eventuality of a sequence having two or more SSRs. The resulting text files contain the full sequence matrix, with all the data from the filtered sequences. The first matrix contain (i) the identifier; (ii) SSR motif type; (iii) SSR motif; (iv) SSR length; (vi) SSR start position; (vii) SSR end position; (viii) sequence length. The second, shorter, matrix is only composed by the (i) new identifier; (ii) sequence identifier; (iii) starting and ending SSR position, and total sequence size (lines 26 and 27). This final file is correctly formatted to be parsed to *splitSSR*, where the SSR is removed from the sequence.

4.2.2 selected_micros

```
1  def selected_micros(rf1, of1, of2):
2      readfile1 = open(rf1, "r")
3      outfile1 = open(of1, "w")
4      outfile2 = open(of2, "w")
5      dic_cluster = {}
6      for line in readfile1:
7          #Split by tab
8          selected_line = line.split("\t")
9          #Creating dictionary for selection of one sequencing per cluster
10         if not selected_line[8] in dic_cluster.keys():
11             dic_cluster[selected_line[8]] = selected_line[0]
12             #Creating outfile
13             sel_micros = list(selected_line[i] for i in [0, 3, 5, 6, 8, 10])
14             outfile2.write("\t".join(sel_micros))
15     #creating tab_selected
16     for keys,values in dic_cluster.items():
17         outfile1.write(values + "\n")
18     readfile1.close()
19     outfile1.close()
20     outfile2.close()
```

Listing 4.14: Procedure for cluster representative selection.

Listing 4.14 shows the function responsible for filtering a single SSR representative from each cluster designated by CD-HIT and its relevant practical information, which will be used to design the primers for each cluster (Figure 3.1–X). In the first part (lines 2–5) it initializes the input and output files and the dictionary *dic cluster* (data structure where all the data will be saved). Next, the first sequence of each cluster is selected and added to the dictionary (lines 6–11). Even if only the first sequence of each cluster is selected, the selection process is in fact randomized, since the operation of FLASH, which occurs at the beginning of the pipeline, causes the random identifier assignment to the sequences, consequently randomizing the sequence order in the input file. Each entry is also accompanied by its respective (i) identifier; (ii) motif type, (iii) SSR start position; (iv) SSR end position, (v) cluster identifier; (vi) cluster size (line 13). Finally, in lines 16 and 17, the dictionary is written into a text file.

4.2.3 allele

```
def allele(rf1,
           of1,
           MIN_ALLELE_CNT,
           MIN_ALLELE_SPECIAL,
           MIN_ALLELE_SPECIAL_DIF):
    readfile1 = open(rf1, "r")
    outfile1 = open(of1, "w")
    cluster_exclude = []
    dic_pool = {}
    dic_allele = {}

    dic_writer(readfile1, dic_pool)
    allele_finder(dic_pool, dic_allele, MIN_ALLELE_SPECIAL)

    if MIN_ALLELE_SPECIAL == 0:
        for cluster in dic_allele.keys():
            if dic_allele[cluster] < MIN_ALLELE_CNT:
                cluster_exclude.append(cluster)
    else:
        for cluster in dic_allele:
            if max(dic_allele[cluster]) - min(dic_allele[cluster]) <
                MIN_ALLEL_SPECIAL_DIF:
                cluster_exclude.append(cluster)
    #Reseting file cursor
    readfile1.seek(0)
    #Selecting sequences from clusters not excluded
    for line in readfile1:
        selected_line = line.split("\t")
        selected_line[9] = selected_line[9].rstrip()
        if selected_line[8] not in cluster_exclude:
            selected_line.append(str(dic_allele[selected_line[8]]) + "\n")
            outfile1.write("\t".join(selected_line))
    readfile1.close()
    outfile1.close()
```

Listing 4.15: Procedure to select SSRs that follow user specification.

The third and final filter, *allele*, will select the loci that have the minimum accepted number of

alleles (Figure 3.1–IX). If "special search" is active, a more loosen analysis is made (Listing 4.15).

At the beginning, all sequence identifiers, with their respective SSR motif type and repetitions, are loaded into the dictionary *dic pool* from the SSR matrix, using the auxiliary function *dic writer* (Listing 4.16). Then, using another auxiliary function called *allele finder* (Listing 4.17), all individual alleles present in each cluster are counted, for being later parsed into the dictionary *dic allele*.

Afterwards, each cluster is filtered. The cluster that do not have the minimum alleles specified by the user are marked for exclusion (lines 15 –22). However, if special search is active a different route is taken. Due to the possible existence of alleles with intermediate number of motifs repetitions of those found in the sample (Section 3.3), the difference between the smallest and highest allele is calculated. Finally, the SSR matrix is again analysed. If the cluster to which the sequence belongs is not marked for exclusion, the sequences, and all their information, are written to the output file (lines 29–33).

```
1  def dic_writer(readfile1, dic):
2      for line in readfile1:
3
4          selected_line = line.split("\t")
5          short_id = selected_line[0][0:10]
6
7          # Writing dictionary
8          if selected_line[8] not in dic.keys():
9              dic[selected_line[8]] = []
10         dic[selected_line[8]].append([short_id, selected_line[3]])
```

Listing 4.16: Auxiliary function of allele detection.

The *dic writer* (Listing 4.16) is a simple function that analyses every SSR from the SSR matrix and stores the necessary attributes for the correct functioning of *allele* (Listing 4.15) in a dictionary, using the cluster identifier as a key and the sequence identifier, SSR motif type and SSR motif repetitions as values.

```
1  def allele_finder(dic_pool, dic_allele, MIN_ALLELE_SPECIAL):
2      for cluster_num in dic_pool:
3          #List of every different allel for a given cluster
4          dic_pool_cluster = {}
5          #Selecting allel size
6          for values in dic_pool[cluster_num]:
7              identifier = values[0]
8              allele_type, allele_size = values[1].split(")")
9              allele_type = allele_type.replace("(","")
10             #adding unique allele to list
11             if allele_size in dic_pool_cluster.keys():
12                 dic_pool_cluster[allele_size][0] += 1
13             else:
14                 dic_pool_cluster[allele_size] = [[],[]]
15                 dic_pool_cluster[allele_size][0] = 1
16             dic_pool_cluster[allele_size][1].append(identifier)
17             dic_allele[cluster_num] = len(dic_pool_cluster.keys())
18             if MIN_ALLELE_SPECIAL == 1:
19                 dic_allele[cluster_num] =
                       list(map(int,dic_pool_cluster.keys()))
```

Listing 4.17: Auxiliary function of allele for allele quantification.

The second auxiliary function of *allele*, *allele finder* will analyze each entry in the dictionary *dic pool*.

All sequences from a given cluster have the same SSR but, due to the pair-end nature of sequencing, the SSR motif can represent forward or reverse sequences. This can cause confusion, since SSRs from the same cluster can have different motifs, but will not affect the overall result. The only thing differentiating the different alleles are the number of motif repetitions.

As such, the number of repetitions is isolated (line 8) and saved (lines 11–16) in a temporary dictionary, *dic pool cluster*. If "special search" is activated, all individual alleles are instead stored in the dictionary (line 23). If not, all different repetitions are stored (line 19). Different repetitions represent different alleles.

4.2.4 python_grep

```
1  def python_grep(rf1, of1):
2      readfile1 = open(rf1, "r")
3      outfile1 = open(of1, "w")
4      for line in readfile1:
5          pattern = r'^GATC\w*GATC$'
6          seq = re.match(pattern, line)
7          if seq:
8              print(re.sub("^@",">",prev_line) + seq.group(0), file=outfile1)
9          prev_line = line
10     readfile1.close()
11   outfile1.close()
```

Listing 4.18: Procedure to select sequences containing the restriction enzyme pattern.

Listing 4.18 describes *python grep* which is the function responsible for selecting only the sequences that contain, in both ends, the restriction enzyme pattern (Figure 3.1 – I). The function employs the usage of regular expressions, which will examine every sequence for the intended pattern. (lines 5 and 6). If the pattern is found, both, sequence and previous line in the file, containing the sequence identifier, are written in the output file. The symbol "@", present in the sequence identifier, is also substituted with ">" to make it suitable for the fasta format.

4.3 File Manipulation

4.3.1 add_id_calculate_length

```
def add_id_calculate_length(rf1, of1, of2):
    readfile1 = open(rf1, "r")
    outfile1 = open(of1, "w")
    outfile2 = open(of2, "w")
    #ID counter
    counter = 1
    #ID saver
    len_line = ""
    # Cycle trough all lines
    for line in readfile1:
        #Selecting only lines with sequence ID
        if line[0] == ">":
            #Adding ID
            ids_line = line[0] + str(counter) + "_" + line[1:]
            #Saving ID string excluding ">"
            len_line = ids_line[1:]
            counter += 1
            outfile1.write(ids_line)
        #Selecting DNA sequence
        else:
            outfile1.write(line)
            #Calculate sequence size
            size = len(line) - 1
            #Constructing final string
            len_line = len_line.strip("\n")
            len_line = len_line + "\t" + str(size)
            outfile2.write(len_line + "\n")
    readfile1.close()
    outfile1.close()
    outfile2.close()
```

Listing 4.19: Procedure to add identifiers and the length calculation

The Listing 4.19 describes the function responsible for calculating the total sequence length, preceding SSR search in the pipeline (Figure 3.1–III).

It takes the output from the Filter 1 and goes trough all lines of the input file. If the line starts with the symbol ">", it contains a sequence identifier. If not, it contains the DNA sequence of the last read sequence identifier. Thus, if the symbol is found (line 12), it copies the entire string ,that is, copies the sequence identifier and adds a new simpler identifier at the start of the already existing sequence identifier (line 14). The simplicity of the new identifier allows for smoother sequence processing. Its attribution is controlled by the variable *counter*, which is incremented each time a new identifier is attributed (line 17).

Since each line is read sequentially, a line containing the sequence identifier is always succeeded by its corresponding DNA sequence. The lack of the symbol ">" means that a DNA sequence was read. In that case, the length of the string is calculated, which corresponds to the DNA sequence size (line 23). With this in mind, the function creates two outputs. The first one, *outfile1*, will contain the newly made sequence identifier followed, in the next line, by the corresponding DNA sequence (lines 14–18). The second output, *outfile2*, will contain the sequence identifier + DNA sequence size, separated by a tab (line 27).

4.3.2 length_merger

```
1  def length_merger(rf1, rf2, of1):
2      readfile1 = open(rf1, "r")
3      readfile2 = open(rf2, "r")
4      outfile1 = open(of1,"w")
5      #Creating dicitonary with SeqID and ssr length
6      dic_length = {}
7      # Dictionary writing
8      for line in readfile1:
9          #Spliting file by tabs
10         selected_line = line.split("\t")
11         #Saving only 10 first caracters of ID.
12         dic_length[selected_line[0][0:10]] = selected_line[1]
13     for line in readfile2:
14         selected_line = line.split("\t")
15         #Removing \n from end tab
16         selected_line[6] = selected_line[6].rstrip()
17         #Creating new tab with ssr length
18         if selected_line[0][0:10] in dic_length.keys():
19             selected_line.append(dic_length[selected_line[0][0:10]])
20             outfile1.write("\t".join(selected_line))
21     readfile1.close()
22     reafile2.close()
23     outfile1.close()
```

Listing 4.20: Procedure to add the length information to the SSR matrix.

length merger (Listing 4.20) is a simple function responsible for pre-processing the input file for the second filter (Figure 3.1–V). It adds the length of the sequence to the SSR matrix (MISA output file), if a SSR was detected. The length was previously calculated with the function at step III of the pipeline by the function *add ids calculate length* (Listing 4.19).

For this, the length calculations are loaded into a dictionary, *dic length* using the sequence identifier as key and length as value (line 12). Next, for each line of the SSR matrix, the sequence identifier is compared to the dictionary entries list. If a match is found, the dictionary values are appended to the line and written in the output file (lines 18–20). If a match is not found, no SSRs were found by MISA and, as such, the sequence and all its information is discarded.

4.3.3 split

```
1  def split(rf1, rf2, of1):
2      readfile1 = open(rf1, "r")
3      readfile2 = open(rf2, "r")
4      outfile1 = open(of1, "w")
5      #Criating Dictionary containing Number ID, Sequence ID, start and end of
           SSR
6      dic_micros = {}
7      for line in readfile1:
8          selected_line = line.split("\t")
9          if len(selected_line) > 1:
10             dic_micros[selected_line[0]] = [selected_line[1][0:10],
                   selected_line[2], selected_line[3]]
11
12     # Search and cut of SSR in selected Sequences
13     ssr_cut(readfile2, outfile1, dic_micros)
14
15     readfile1.close()
16     readfile2.close()
17     outfile1.close()
```

Listing 4.21: Procedure to remove SSRs from respective sequence.

The function *split* (Listing 4.21) removes the SSR from the DNA sequence, thus, the SSRs can be clustered not by motif structure, but by its flanking regions, avoiding errors due to homoplasy (Section 1.2.4.3). Two inputs will be parsed to the function: the second filter result (Figure 3.1–V) and elements containing the sequence identifier and corresponding DNA strings (Figure 3.1–I).

Whith this inputs, a dictionary is created, *dic_micros* saving the secondary identifier (created for the possible existence of multiple SSRs in the same sequence) as a key together with (i) the sequence identifier; (ii) SSR start position; (iii) SSR end position as values(lines 7 – 10). This data is parsed to the auxiliary function *ssr_cut* (Listing 4.22) that slices the SSR and writes the sequence without the SSR to the output.

```
1  def ssr_cut(readfile2, outfile1, dic):
2      last_seq = ""
3      dna_line = ""
4      for line in readfile2:
5          selected_line = line.split("\t")
6          for key in dic:
7              if selected_line[0][1:11] in str(dic[key][0]):
8                  #Selecting Sequence ID
9                  if selected_line[0][0] == ">":
10                     #If there is multiple SSRs the DNA Seq is saved
11                     if last_seq == selected_line[0][1:11]:
12                         nextline = dna_line
13                         last_seq = ""
14                     else:
15                         #Selection of the next line, containing the DNA
                               sequence
16                         nextline = readfile2.readline()
17                         dna_line = nextline
18                     #Slicing away SSR
19                     first_cut = int(dic.get(key)[1]) - 1
20                     second_cut = int(dic.get(key)[2])
21                     nextline = nextline[0 : first_cut] + nextline[second_cut: ]
22                     #Output writing
23                     outfile1.write(selected_line[0] + nextline)
24                 #Saving last sequence ID:
25                 if selected_line[0][0] == ">":
26                     last_seq = selected_line[0][1:11]
```

Listing 4.22: Auxiliary function of split.

Each sequence on the second input file is analyzed and checked for an entry on the dictionary calculated in *split* function. If an entry exists, the script checks for the existence of multiple SSRs in the same sequence. If, by chance, two identical sequence identifiers are read consecutively, it is certain that the DNA sequence contains two or more SSRs. Because of this, the script compares the current sequence identifier with the last sequence identifier cycled through. If no correspondence is found the next line (containing the DNA sequence) is read. If the opposite occurs, the DNA sequence from the last sequence cycled, saved in *last_seq*, is read(lines 14 – 17). Finally, the sequence is spliced from its SSR using the data present in the dictionary. The combined sequence identifier plus the sequence without SSR is written in the output file.

4.3.4 cluster

```
1  def cluster(rf1, of1):
2      readfile1 = open(rf1, "r")
3      outfile1 = open(of1, "w")
4      current_cluster = ""
5      #Creating dictionary to save each sequence of each cluster
6      dic_cluster = {}
7      for line in readfile1:
8          #Selecting cluster ID
9          if ">" in line[0]:
10             selected_line = line.split()
11             current_cluster = selected_line[1]
12             #Creating cluster entry on the dictionary
13             dic_cluster[current_cluster] = []
14         else:
15             #Replacing "_" for ">"" for easy selection of sequence ID
16             line = line.replace("_", ">")
17             selected_line = line.split(">")
18             #Adding sequence ID to corresponding cluster on the dictionary
19             dic_cluster[current_cluster].append(selected_line[1])
20     for key, value in dic_cluster.items():
21         for x in value:
22             #Defining cluster size
23             size_cluster = len(dic_cluster[key])
24             outfile1.write(str(x) + "\t" + str(key) + "\t" + str(size_cluster)
                   + "\n")
25     readfile1.close()
26     outfile1.close()
```

Listing 4.23: Procedure to export data from CD-HIT.

Listing 4.23 details the actions performed by the *cluster* function, one of two functions responsible for step VIII in the pipeline (Figure 3.1). It takes the cluster data from CD-HIT (Figure 3.1–VII) and creates a more easy to handle and readable output, *cluster_out*, detailing for each sequence identifier its asigned cluster and cluster size.

Each line of the input file is analyzed. The cluster identifier can be identified by the presence of the symbol ">" at the start of the string. This cluster identifier is used as a key in *dic cluster*, a dictionary created for the effect (line 13). In the CD-HIT output format, each cluster identifier

line is followed by the sequence identifiers of that cluster. Thus, each adjacent line is added as a value to the dictionary entry of the current cluster (line 19). When a new cluster identifier is detected, the cycle starts again the procedure. At the end, the number of values for each entry in *dic cluster* is measured. Finally, all sequences are written in the output file, with the respective cluster number and cluster size.

4.3.5 add_cluster_info

```
1  def add_cluster_info(rf1, rf2, of1):
2      readfile1 = open(rf1, "r")
3      readfile2 = open(rf2, "r")
4      outfile1 = open(of1, "w")
5      #Creating dictionary to save cluster # and cluster size
6      dic_cluster = {}
7      for line in readfile1:
8          selected_line = line.split("\t")
9          selected_line[2] = selected_line[2].rstrip()
10         #Create dictionary using ID as key and cluster # and cluster size
11         dic_cluster[selected_line[0]] =  [selected_line[1], selected_line[2]]
12     for line in readfile2:
13         # Split by tab and remove \n
14         selected_line = line.rstrip()
15         selected_line = selected_line.split("\t")
16         #Selecting only ID
17         ID, junk = selected_line[0].split("_")
18         #Appending to misa file cluster # and size
19         for item in dic_cluster[ID]:
20             selected_line.append(item)
21         #Outfile writing
22         outfile1.write("\t".join(selected_line) + "\n")
23     reafile1.close()
24     readfile2.close()
25     outfile1.close()
```

Listing 4.24: Procedure to insert the cluster information in the SSR Matrix.

The function *add cluster info* (Listing 4.24), is the second function for clustering the SSR sequences (Figure 3.1–VIII). This simple function goal is to add cluster information obtained in the function *cluster* (Listing 4.23) to the general SSR matrix, where various SSR parameters are

stored. Basically, the dictionary *dic cluster* is initiated with the output from *cluster*, using the sequence identifier as key and cluster identifier and cluster size as values. This information is added to the SSR matrix by comparing each sequence identifier with its corresponding entry in the dictionary (line 19 and 20).

4.4 Output Preparation

4.4.1 pseudofasta

```
1  def pseudofasta(rf1, rf2, of1):
2      readfile1 = open(rf1, "r")
3      readfile2 = open(rf2, "r")
4      outfile1 = open(of1, "w")
5      selected_seqs = []
6      for line in readfile1:
7          selected_seqs.append(line)
8      for line in readfile2:
9          if line[0] == ">" :
10             if line[1:] in selected_seqs:
11                 nextline = readfile2.readline()
12                 outfile1.write("SEQUENCE_ID=" + line[1:] +
                       "SEQUENCE_TEMPLATE=" + nextline + "=" + "\n")
13     reafile1.close()
14     readfile2.close()
15     outfile1.close()
```

Listing 4.25: Procedure to create the Primer3 input file.

The input file for Primer3 must follow a specific format for its correct functioning. Thus, the chosen cluster representatives sequences must be converted by *pseudofasta* (Listing 4.25) before being processed by Primer3 (Figure 3.1–X). All sequence identifiers from the cluster representatives are saved to a temporary list *selected seqs* (lines 8 and 7). Afterwards, each line of the input file, containing the original non altered sequences with both SSR and flanking regions, are inspected, line by line. If the sequence identifier is present in *selected seqs*, the sequence identifier, followed by the corresponding sequence in the adjacent line, is written in the correct Primer3 format (line 12). This cycle runs till every sequence in the file is analyzed.

4.4.2 final_primers

```
def final_primers(rf1, rf2, of1):
    readfile1 = open(rf1, "r")
    readfile2 = open(rf2, "r")
    outfile1 = open(of1, "w")
    dic_attr = {}
    for line in readfile1:
        line = line.rstrip()
        selected_line = line.split("\t")
        dic_attr[selected_line[0]] = [selected_line[1], selected_line[2],
            selected_line[3], selected_line[4], selected_line[5]]
    transform(readfile2, outfile1, dic_attr)
    readfile1.close()
    readfile2.close()
    outfile1.close()
```

Listing 4.26: Procedure to convert Primer3 result to final pipeline result.

The output from Primer3 contains all the information needed for amplification of the selected SSRs, such as the generated forward and reverse primer and respective melting temperature (Tm), primer product size, among others. Despite this, the format is not easily analysed or readable, due to its code based format. To improve user usability and convenience, *final primers* converts Primer3 output to the final output of the pipeline (listing 4.26). Firstly, data of the start and end of the SSR plus number of alleles of the respective SSR loci are loaded into the dictionary *dic attr*. This data is then parsed into the auxiliary function *transform* that will create the final result.

```
def transform(readfile, outfile, dic):
    out = []
    for line in readfile:
        line = line.rstrip()
        if re.match("^SEQUENCE_ID", line):
            seq_id = line.split("=")[1]
            count = 0
        elif re.match("^PRIMER_LEFT_\d_SEQUENCE", line):
            out.append(line.split("=")[1])
        elif re.match("^PRIMER_RIGHT_\d_SEQUENCE", line):
            out.append(line.split("=")[1])
        elif re.match("^PRIMER_LEFT_\d_TM", line):
            out.append(line.split("=")[1])
        elif re.match("^PRIMER_RIGHT_\d_TM=", line):
            out.append(line.split("=")[1])
        elif re.match("^PRIMER_LEFT_\d=(\d*),(\d*)", line):
            next = line.split("=")[1]
            left_ini, left_len = next.split(",")
        elif re.match("^PRIMER_RIGHT_\d=(\d*),(\d*)", line):
            next = line.split("=")[1]
            right_ini, right_len = next.split(",")
        elif re.match("^PRIMER_PAIR_\d_PRODUCT_SIZE=\d", line):
            out.append(line.split("=")[1])
            out.append(dic[seq_id][0])
            out.append(dic[seq_id][4])
            start = int(dic[seq_id][1])
            end = int(dic[seq_id][2])
            cluster = dic[seq_id][3]
            good_left = int(left_ini) + int(left_len)
            good_right = int(right_ini) - int(right_len)
            if (good_left < start) and (good_right > end):
                out = list(out[i] for i in [4,0,2,1,3,5,6])
                out = [str(seq_id)] + out
                outfile.write("\t".join(out))
                if count == 0 :
                    outfile.write("\t| BEST |" )
                    count  = 1
                outfile.write("\n")
            out = []
```

Listing 4.27: Procedure to show final results.

The auxiliary function *transform* reads the Primer3 output and transforms it in to the new format (Listing 4.27). Resorting to regular expressions, various attributes are searched for each individual SSR primer pair created (lines 4–22) and added to *out*, which will save all information for every single primer for later be written. It will be also verified if the generated primer does not exceed SSR flanking regions (line 31). The final output file is divided by tabs and contains eight columns (Figure 4.1). Each line represents the primers designed by Primer3 for each SSR found in the sample. (a) First column (red) contains the sequence identifier. For each loci representative various primers can be designed, giving the users various options for optimal amplification. (b) Next, the total size of the SSR is shown (Brown). The primers designed are shown next, starting with the left primer and respective Tm, followed by the right Primer and respective Tm. Both primers are shown with direction 5' to 3'. Then, the specific motif/allele found is shown with the total number of alleles for the specific SSR loci (black). The flag "|BEST|" is shown if Primer3 defines the corresponding primer as the best one for loci amplification.

Sequence ID
5_HWI-M01998:26:000000000-D2MKR:1:1101:11464:2210 1:N:0:CAGATC
5_HWI-M01998:26:000000000-D2MKR:1:1101:11464:2210 1:N:0:CAGATC
5_HWI-M01998:26:000000000-D2MKR:1:1101:11464:2210 1:N:0:CAGATC
5_HWI-M01998:26:000000000-D2MKR:1:1101:11464:2210 1:N:0:CAGATC
5_HWI-M01998:26:000000000-D2MKR:1:1101:11464:2210 1:N:0:CAGATC
27_HWI-M01998:26:000000000-D2MKR:1:1101:14803:1870 1:N:0:CAGATC
27_HWI-M01998:26:000000000-D2MKR:1:1101:14803:1870 1:N:0:CAGATC
27_HWI-M01998:26:000000000-D2MKR:1:1101:14803:1870 1:N:0:CAGATC
27_HWI-M01998:26:000000000-D2MKR:1:1101:14803:1870 1:N:0:CAGATC
27_HWI-M01998:26:000000000-D2MKR:1:1101:14803:1870 1:N:0:CAGATC
45_HWI-M01998:26:000000000-D2MKR:1:1101:12042:2591 1:N:0:CAGATC
164_HWI-M01998:26:000000000-D2MKR:1:1101:16498:4006 1:N:0:CAGATC
164_HWI-M01998:26:000000000-D2MKR:1:1101:16498:4006 1:N:0:CAGATC
164_HWI-M01998:26:000000000-D2MKR:1:1101:16498:4006 1:N:0:CAGATC
164_HWI-M01998:26:000000000-D2MKR:1:1101:16498:4006 1:N:0:CAGATC
164_HWI-M01998:26:000000000-D2MKR:1:1101:16498:4006 1:N:0:CAGATC
339_HWI-M01998:26:000000000-D2MKR:1:1101:14970:5186 1:N:0:CAGATC
339_HWI-M01998:26:000000000-D2MKR:1:1101:14970:5186 1:N:0:CAGATC
339_HWI-M01998:26:000000000-D2MKR:1:1101:14970:5186 1:N:0:CAGATC

(a)

Size	Left Primer	Tm	Right Primer	Tm	Motif/Allleles	Flag
146	TCAATTGGCAAGATTTGCTTC	60.203	GTATGGAAGCATCATAGGTGTCC	59.744	(AC)22 16	| BEST |
147	TCAATTGGCAAGATTTGCTTC	60.203	GGTATGGAAGCATCATAGGTGTC	59.744	(AC)22 16	
151	TCAATTGGCAAGATTTGCTTC	60.203	TCAAGGTATGGAAGCATCATAGG	60.338	(AC)22 16	
158	CATGTTGTTTCAATTGGCAAG	59.073	AAGGTATGGAAGCATCATAGGTG	59.407	(AC)22 16	
148	TCAATTGGCAAGATTTGCTTC	60.203	AGGTATGGAAGCATCATAGGTGT	60.616	(AC)22 16	
214	ATGTGCACCCTGATGTTCAC	59.399	CATCCATGTTGTTGGAATGG	59.623	(GT)12 8	| BEST |
220	AAAGAAATGTGCACCCTGATG	59.985	CATCCATGTTGTTGGAATGG	59.623	(GT)12 8	
213	GTGCACCCTGATGTTCACTG	60.162	ACATCCATGTTGTTGGAATGG	60.489	(GT)12 8	
211	TGCACCCTGATGTTCACTG	59.196	CATCCATGTTGTTGGAATGG	59.623	(GT)12 8	
215	GTGCACCCTGATGTTCACTG	60.162	GAACATCCATGTTGTTGGAATG	60.096	(GT)12 8	
211	CACAACCAGGATATTGACGATG	60.249	AAGATTTCCACATGCCACTTG	59.985	(GT)12 14	| BEST |
214	ATGTGCACCCTGATGTTCAC	59.399	CATCCATGTTGTTGGAATGG	59.623	(GT)8 7	| BEST |
220	AAAGAAATGTGCACCCTGATG	59.985	CATCCATGTTGTTGGAATGG	59.623	(GT)8 7	
213	GTGCACCCTGATGTTCACTG	60.162	ACATCCATGTTGTTGGAATGG	60.489	(GT)8 7	
211	TGCACCCTGATGTTCACTG	59.196	CATCCATGTTGTTGGAATGG	59.623	(GT)8 7	
215	GTGCACCCTGATGTTCACTG	60.162	GAACATCCATGTTGTTGGAATG	60.096	(GT)8 7	
262	CTGGTCACAAAGGAGAAATGG	59.578	TGAACACTCTCCCAAATGGTC	59.958	(CT)28 14	| BEST |
255	CTGGTCACAAAGGAGAAATGG	59.578	TCTCCCAAATGGTCACTGG	59.457	(CT)28 14	
260	GGTCACAAAGGAGAAATGGAAG	59.976	TGAACACTCTCCCAAATGGTC	59.958	(CT)28 14	

(b)

Figure 4.1: Example of the Micro-Primers output.

Chapter 5

Results

The Micro-Primers pipeline was tested with a dataset belonging to bats of the species *Hiposideros vittatus* from two different populations, Namibia and Botswana, with a total of 15 and 21 individuals[1], respectively.

Samples were mixed into a unique library and later sequenced on a Illumina MySeq producing pair-end reads of 251 nucleotides length. The library preparation was made following the protocol established at [78] and explained at the Input files section.

Since the species is diploid, the maximum number of alleles to be found by locus could reach the number of 72. The process was reproduced both, by running the programs sequentially by hand, including some files pre-processing so that certain outputs are suited to be parsed to other program, and executing the Micro-Primers pipeline with exactly the same parameters used in the manual run.

Final results in both processes were identical, as expected, being the only difference the time spent to reach the final result. The manual process took not less than 24 hours, mainly invested on the clusters eye selection. Some format changes were needed as well for the proper functioning of certain programs, such as Primer3. Sequences identifiers also had to be changed for easier handling, allowing for a selection based on a unique ID and not on a long identifier, only different at the end of a very long name string.

On the other hand, the automatic pipeline took less than 120 seconds to perform the analysis on a Intel i7 Octacore desktop with 64 Gb of memory. It should be noticed that the whole

[1]This dataset is included in the Micro-Primers' GitHub repository.

process was run on a single core and the unique point where the memory is demanding is at the trimming step carried out by Trimmomatic, so minimal resources are needed in general. Three different parameters combinations were tried to check differences in the number of microsatellites loci selected. The number of sequences remaining after every stage of the pipeline were calculated and are shown at Table 5.1.

The pipeline execution was modified by changing the parameters at the config file, in the SSR section. The three configurations used were:

- MIN_ALLEL_CNT=5, MIN_ALLEL_SPECIAL=0 (default).
- MIN_ALLEL_CNT=4, MIN_ALLEL_SPECIAL=0 (4 unique alleles).
- MIN_ALLEL_SPECIAL=1, MIN_ALLEL_SPECIAL_DIFF=8 (difference of 8).

The number of selected sequences at each step remains the same between the three experiments until the cluster selection, where the number of alleles makes the difference. Across all steps there is a significant reduction in the sequences number, with an important drop at the filter steps.

Analyzing the experimental results, it is clear that the activation of the SPECIAL parameter produces an increase in the number of results. However, these new results will be of a lower quality, since any cluster with any number of alleles will be accepted up till the difference between the maximum number of repeats and the minimal is over the established in the config file.

If we observe the number of alleles in the final results at the configuration "5-0" the distribution of the selected clusters is 16,14,14,12,10,8,8,7,7,7,6,5,5,5,5,5,5. In the configuration "4-0" the distribution remains the same with eighteen additional clusters with 4 alleles each. Meanwhile at the "5-1-8" configuration, 38 new clusters appear, including fifteen clusters with 3 alleles and twenty three with only two alleles, all of them with a range difference of 8.

During the analysis, the number of individual alleles for each cluster detected can be huge, but only a small part of them represent informative alleles, that is, only sequences with different number of repeats of the motif will be considered as alleles, and all the sequences with the same number of repeats will be ignored for computing the number of final alleles. This process is carried out automatically by Micro-Primers but on a manual search for microsattelites all this remaining information can represent a lot of effort to discriminate them or a lot of spent money in failed amplifications that will not result of interest.

Table 5.1: Number of sequences being kept at every pipeline step. Numbers for the three configurations are equal until clusters creation. The relative percentage over the original is shown in brackets.

Pipeline Step	Combinations		
	5-0	4-0	5-1-8
FASTQ File	259506 (100%)		
Trimming	188843 (72.77%)		
Pair-end Merge	130603 (50.32%		
Filter 1	19695 (7.59%)		
With SSRs	16983 (6.54%)		
Filter 2	8083 (3.11%)		
Filter 3	2091 (0.81%)	2204 (0.84%)	4913 (1.89%)
Unique Alleles	26 (0.01%)	51 (0.02%)	104 (0.04%)
Primers Selected	17 (0.007%)	35 (0.013%)	73 (0.028%)

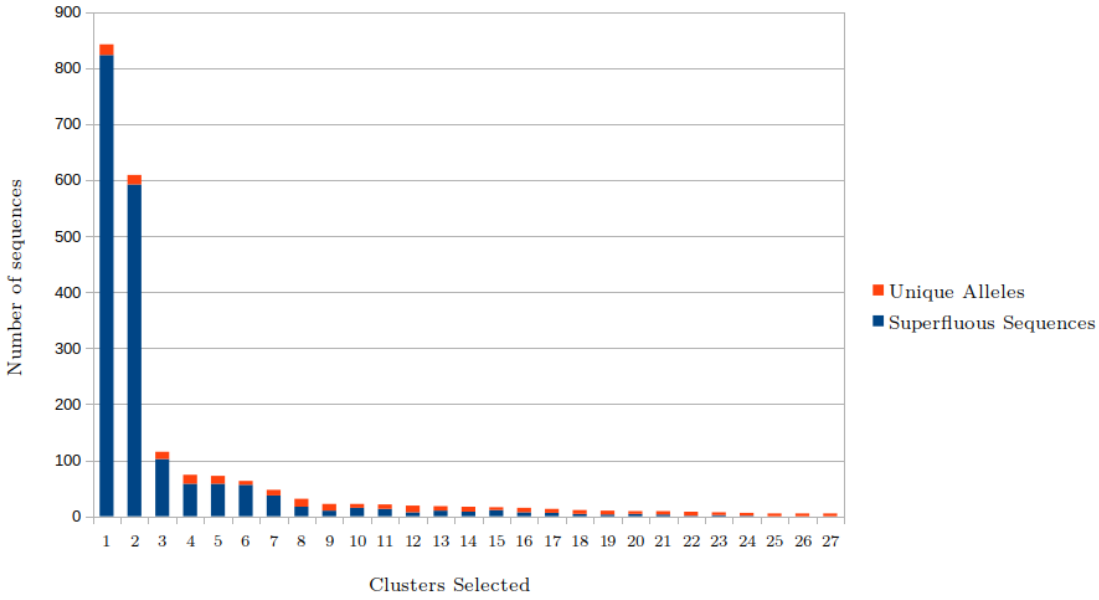

Figure 5.1: Total number versus unique alleles

Using the same data as before, the total number of unique sequences in the selected 27 clusters was 2091. Among those, only 243 sequences contained unique alleles, a ratio of only 11.63 percent (Figure 5.1).

Chapter 6

Discussion and Conclusions

Nowadays, it can take up to months for researches to analyse and select SSR primers for their projects. It requires them to run MISA several times without any kind of pre-processing and to select by hand individual candidates for primer3. With no clusterisation or ignoring the number of available alleles, most amplifications results are useless since most of the time SSRs are not variable across the population. This fact implies a waste of time and money that could be easily avoided just with the development of an automatic tool to do the selection based on polymorphism. In that way a user can change parameters several times and select only that SSRs most suitable for the experiment just in few seconds.

Comparison between the manual selection and the execution of Micro-Primers is very biased towards the automatic pipeline since the process by hand still needs some scripting or command line pre-processing between software execution just to adapt the output of one program to the input of the next. The longest process in the selection corresponds to the categorization and allele counting for every single SSR. First, all microsatellites individually identified need to be grouped by locus taking into account that from sequencing we will have both strands of the same sequence so the same locus can be represented by two different sequences as well as the microsatellites. After the identification of all the copies of the same microsatellite loci one needs to count the number of different unique alleles for subsequent selection. In some cases this task can result very tedious since some of the locus can be represented by thousands of elements. The creation of an automatic pipeline that can speed up the process immune to manual selection bias is of a great interest for the scientific community.

The use of Micro-Primers provides an unprecedented availability of candidate SSRs at a

much more reliable and fast pace than before. This abundance permits selection of markers that may be most suitable for specific applications or particular organisms.

By making the process automatic, a researcher can run several times the same dataset just modifying the parameters in the config file to get different results. As results showed above, the inclusion of the SPECIAL parameter makes a wider selection, where the researcher accepts the hypothesis of the existence of new alleles in the species between the maximum and minimum found in the dataset, due maybe to a insufficient sequencing.

Micro-Primers is a novel tool with space for improvement. The creation of python implementations of the third-party software included in this pipeline can lead to total compatibility with Windows operating systems and, in combination with pipeline parallelization, can greatly improve performance and lower pipeline run time. Also, regarding data quality, more options can be added to extend the functionality and its malleability to different samples. The addition of a Graphic Use Interface can facilitate the user configuration and runtime, avoiding the need to alter text-based files.

Concluding, Micro-Primers has demonstrated an enormous potential for microsatellite search and primer design among other similar software since it includes an important step of selection of the more polymorphic ones without any user supervision. The complete automation of the process make it very easy to use for any researcher, even with no experience at all in command-line software thanks to the inclusion of a Python script that will install all the required software and dependencies before the pipeline execution. Micro-Primers allows any researcher to characterize individuals or a whole population just with some PCR reactions. This protocol can be of an great help for conservation plans where the species are becoming very homogeneous and a cross-population hybridization is needed.

Bibliography

[1] J. M. Heather and B. Chain, "The sequence of sequencers: The history of sequencing dna," *Genomics*, vol. 107, no. 1, pp. 1–8, 2016.

[2] F. Sanger, S. Nicklen, and A. R. Coulson, "Dna sequencing with chain-terminating inhibitors," *Proceedings of the national academy of sciences*, vol. 74, no. 12, pp. 5463–5467, 1977.

[3] W. Ansorge, B. S. Sproat, J. Stegemann, and C. Schwager, "A non-radioactive automated method for dna sequence determination," *Journal of Biochemical and Biophysical Methods*, vol. 13, no. 6, pp. 315–323, 1986.

[4] H. Swerdlow and R. Gesteland, "Capillary gel electrophoresis for rapid, high resolution dna sequencing," *Nucleic acids research*, vol. 18, no. 6, pp. 1415–1419, 1990.

[5] F. R. Blattner, G. Plunkett, C. A. Bloch, N. T. Perna, V. Burland, M. Riley, J. Collado-Vides, J. D. Glasner, C. K. Rode, G. F. Mayhew, *et al.*, "The complete genome sequence of escherichia coli k-12," *science*, vol. 277, no. 5331, pp. 1453–1462, 1997.

[6] A. G. Initiative *et al.*, "Analysis of the genome sequence of the flowering plant arabidopsis thaliana," *nature*, vol. 408, no. 6814, p. 796, 2000.

[7] M. D. Adams, S. E. Celniker, R. A. Holt, C. A. Evans, J. D. Gocayne, P. G. Amanatides, S. E. Scherer, P. W. Li, R. A. Hoskins, R. F. Galle, *et al.*, "The genome sequence of drosophila melanogaster," *Science*, vol. 287, no. 5461, pp. 2185–2195, 2000.

[8] I. H. G. S. Consortium *et al.*, "Finishing the euchromatic sequence of the human genome," *Nature*, vol. 431, no. 7011, p. 931, 2004.

[9] J. C. Venter, M. D. Adams, E. W. Myers, P. W. Li, R. J. Mural, G. G. Sutton, H. O. Smith, M. Yandell, C. A. Evans, R. A. Holt, *et al.*, "The sequence of the human genome," *science*, vol. 291, no. 5507, pp. 1304–1351, 2001.

[10] A. Rohlin, J. Wernersson, Y. Engwall, L. Wiklund, J. Björk, and M. Nordling, "Parallel sequencing used in detection of mosaic mutations: comparison with four diagnostic dna screening techniques," *Human mutation*, vol. 30, no. 6, pp. 1012–1020, 2009.

[11] Ş. Ari and M. Arikan, "Next-generation sequencing: advantages, disadvantages, and future," in *Plant Omics: Trends and Applications*, pp. 109–135, Springer, 2016.

[12] M. Margulies, M. Egholm, W. E. Altman, S. Attiya, J. S. Bader, L. A. Bemben, J. Berka, M. S. Braverman, Y.-J. Chen, Z. Chen, *et al.*, "Genome sequencing in microfabricated high-density picolitre reactors," *Nature*, vol. 437, no. 7057, p. 376, 2005.

[13] I. Illumina, "An introduction to next-generation sequencing technology," 2015.

[14] J. M. Rothberg and J. H. Leamon, "The development and impact of 454 sequencing," *Nature biotechnology*, vol. 26, no. 10, p. 1117, 2008.

[15] C. Luo, D. Tsementzi, N. Kyrpides, T. Read, and K. T. Konstantinidis, "Direct comparisons of illumina vs. roche 454 sequencing technologies on the same microbial community dna sample," *PloS one*, vol. 7, no. 2, p. e30087, 2012.

[16] M. L. Metzker, "Emerging technologies in dna sequencing," *Genome research*, vol. 15, no. 12, pp. 1767–1776, 2005.

[17] J. M. Rothberg, W. Hinz, T. M. Rearick, J. Schultz, W. Mileski, M. Davey, J. H. Leamon, K. Johnson, M. J. Milgrew, M. Edwards, *et al.*, "An integrated semiconductor device enabling non-optical genome sequencing," *Nature*, vol. 475, no. 7356, p. 348, 2011.

[18] J. Perkel, "Making contact with sequencing's fourth generation," *BioTechniques*, vol. 50, no. 2, pp. 93–95, 2011.

[19] C. Bleidorn, "Third generation sequencing: technology and its potential impact on evolutionary biodiversity research," *Systematics and biodiversity*, vol. 14, no. 1, pp. 1–8, 2016.

[20] M. L. C. Vieira, L. Santini, A. L. Diniz, and C. d. F. Munhoz, "Microsatellite markers: what they mean and why they are so useful," *Genetics and molecular biology*, vol. 39, no. 3, pp. 312–328, 2016.

[21] K. J. Verstrepen, A. Jansen, F. Lewitter, and G. R. Fink, "Intragenic tandem repeats generate functional variability," *Nature genetics*, vol. 37, no. 9, p. 986, 2005.

[22] B. Brinkmann, M. Klintschar, F. Neuhuber, J. Hühne, and B. Rolf, "Mutation rate in human microsatellites: influence of the structure and length of the tandem repeat," *The American Journal of Human Genetics*, vol. 62, no. 6, pp. 1408–1415, 1998.

[23] W. Amos, "Heterozygosity increases microsatellite mutation rate," *Biology letters*, vol. 12, no. 1, p. 20150929, 2016.

[24] E. J. Oliveira, J. G. Pádua, M. I. Zucchi, R. Vencovsky, and M. L. C. Vieira, "Origin, evolution and genome distribution of microsatellites," *Genetics and Molecular Biology*, vol. 29, no. 2, pp. 294–307, 2006.

[25] Z. Wang, J. Weber, G. Zhong, and S. Tanksley, "Survey of short tandem dna repeats," *TAG. Theoretical and applied genetics. Theoretische und angewandte Genetik*, vol. 88, pp. 1–6, 04 1994.

[26] Y. Edwards, G. Elgar, M. Clark, and M. Bishop, "The identification and characterization of microsatellites in the compact genome of the japanese pufferfish, fugu rubripes: Perspectives in functional and comparative genomic analyses," *Journal of molecular biology*, vol. 278, pp. 843–54, 06 1998.

[27] M. Morgante, M. Hanafey, and W. Powell, "Microsatellites are preferentially associated with nonrepetitive dna in plant genomes," *Nature genetics*, vol. 30, no. 2, p. 194, 2002.

[28] D. Metzgar, J. Bytof, and C. Wills, "Selection against frameshift mutations limits microsatellite expansion in coding dna," *Genome research*, vol. 10, no. 1, pp. 72–80, 2000.

[29] A. Bhargava and F. Fuentes, "Mutational dynamics of microsatellites," *Molecular biotechnology*, vol. 44, no. 3, pp. 250–266, 2010.

[30] J. L. Weber and C. Wong, "Mutation of human short tandem repeats," *Human molecular genetics*, vol. 2, no. 8, pp. 1123–1128, 1993.

[31] E. M. Black and S. Giunta, "Repetitive fragile sites: centromere satellite dna as a source of genome instability in human diseases," *Genes*, vol. 9, no. 12, p. 615, 2018.

[32] A. J. Verkerk, M. Pieretti, J. S. Sutcliffe, Y.-H. Fu, D. P. Kuhl, A. Pizzuti, O. Reiner, S. Richards, M. F. Victoria, F. Zhang, *et al.*, "Identification of a gene (fmr-1) containing a cgg repeat coincident with a breakpoint cluster region exhibiting length variation in fragile x syndrome," *Cell*, vol. 65, no. 5, pp. 905–914, 1991.

[33] J. D. Brook, M. E. McCurrach, H. G. Harley, A. J. Buckler, D. Church, H. Aburatani, K. Hunter, V. P. Stanton, J.-P. Thirion, T. Hudson, *et al.*, "Molecular basis of myotonic dystrophy: expansion of a trinucleotide (ctg) repeat at the 3 end of a transcript encoding a protein kinase family member," *Cell*, vol. 68, no. 4, pp. 799–808, 1992.

[34] G.-F. Richard and F. Pâques, "Mini-and microsatellite expansions: the recombination connection," *EMBO reports*, vol. 1, no. 2, pp. 122–126, 2000.

[35] H. Debrauwere, C. Gendrel, S. Lechat, and M. Dutreix, "Differences and similarities between various tandem repeat sequences: minisatellites and microsatellites," *Biochimie*, vol. 79, no. 9-10, pp. 577–586, 1997.

[36] B. A. Payseur and M. W. Nachman, "Microsatellite variation and recombination rate in the human genome," *Genetics*, vol. 156, no. 3, pp. 1285–1298, 2000.

[37] D. Bachtrog, S. Weiss, B. Zangerl, G. Brem, and C. Schlötterer, "Distribution of dinucleotide microsatellites in the drosophila melanogaster genome.," *Molecular Biology and Evolution*, vol. 16, no. 5, pp. 602–610, 1999.

[38] A. J. Jeffreys, R. Neumann, M. Panayi, S. Myers, and P. Donnelly, "Human recombination hot spots hidden in regions of strong marker association," *Nature genetics*, vol. 37, no. 6, p. 601, 2005.

[39] J. Majewski and J. Ott, "Gt repeats are associated with recombination on human chromosome 22," *Genome research*, vol. 10, no. 8, pp. 1108–1114, 2000.

[40] V. I. Hashem, W. A. Rosche, and R. R. Sinden, "Genetic recombination destabilizes (ctg) n·(cag) n repeats in e. coli," *Mutation Research/Fundamental and Molecular Mechanisms of Mutagenesis*, vol. 554, no. 1-2, pp. 95–109, 2004.

[41] Y. ZHU, J. E. STRASSMANN, and D. C. QUELLER, "Insertions, substitutions, and the origin of microsatellites," *Genetics Research*, vol. 76, no. 3, pp. 227–236, 2000.

[42] W. Messier, S.-H. Li, and C.-B. Stewart, "The birth of microsatellites," *Nature*, vol. 381, no. 6582, p. 483, 1996.

[43] O. Rose and D. Falush, "A threshold size for microsatellite expansion.," *Molecular biology and evolution*, vol. 15, no. 5, pp. 613–615, 1998.

[44] D. Dieringer and C. Schlötterer, "Two distinct modes of microsatellite mutation processes: evidence from the complete genomic sequences of nine species," *Genome research*, vol. 13, no. 10, pp. 2242–2251, 2003.

[45] S. Ohno, "So much'junk'dna in our genome," in *Evolution of Genetic Systems, Brookhaven Symp. Biol.*, pp. 366–370, 1972.

[46] N. Horie, H. Aiba, K. Oguro, H. Hojo, and K. Takeishi, "Functional analysis and dna polymorphism of the tandemly repeated sequences in the 5'-terminal regulatory region of the human gene for thymidylate synthase," *Cell structure and function*, vol. 20, no. 3, pp. 191–197, 1995.

[47] M. J. Lawson and L. Zhang, "Housekeeping and tissue-specific genes differ in simple sequence repeats in the 5'-utr region," *Gene*, vol. 407, no. 1-2, pp. 54–62, 2008.

[48] Y.-C. Li, A. B. Korol, T. Fahima, and E. Nevo, "Microsatellites within genes: structure, function, and evolution," *Molecular biology and evolution*, vol. 21, no. 6, pp. 991–1007, 2004.

[49] C. Gao, X. Ren, A. S. Mason, J. Li, W. Wang, M. Xiao, and D. Fu, "Revisiting an important component of plant genomes: microsatellites," *Functional Plant Biology*, vol. 40, no. 7, pp. 645–661, 2013.

[50] E. Buschiazzo and N. J. Gemmell, "The rise, fall and renaissance of microsatellites in eukaryotic genomes," *Bioessays*, vol. 28, no. 10, pp. 1040–1050, 2006.

[51] H. H. Kazazian, "Mobile elements: drivers of genome evolution," *science*, vol. 303, no. 5664, pp. 1626–1632, 2004.

[52] E. Nadir, H. Margalit, T. Gallily, and S. A. Ben-Sasson, "Microsatellite spreading in the human genome: evolutionary mechanisms and structural implications," *Proceedings of the National Academy of Sciences*, vol. 93, no. 13, pp. 6470–6475, 1996.

[53] J. S. Taylor, J. Durkin, and F. Breden, "The death of a microsatellite: a phylogenetic perspective on microsatellite interruptions.," *Molecular Biology and Evolution*, vol. 16, no. 4, pp. 567–572, 1999.

[54] W. Powell, G. C. Machray, and J. Provan, "Polymorphism revealed by simple sequence repeats," *Trends in plant science*, vol. 1, no. 7, pp. 215–222, 1996.

[55] D. L. Rimoin, J. M. Connor, R. E. Pyeritz, and B. R. Korf, *Emery and Rimoin's principles and practice of medical genetics.* Churchill Livingstone Elsevier, 2007.

[56] S. Poompuang and E. M. Hallerman, "Toward detection of quantitative trait loci and marker-assisted selection in fish," *Reviews in Fisheries Science*, vol. 5, no. 3, pp. 253–277, 1997.

[57] C. BC and M. DJ, "Marker-assisted selection: an approach for precision plant breeding in the twenty-first century.," *Philos Trans R Soc Lond B Biol Sci*, vol. 363, no. 1491, pp. 557–572, 2008.

[58] D. A. Chistiakov, B. Hellemans, and F. A. Volckaert, "Microsatellites and their genomic distribution, evolution, function and applications: a review with special reference to fish genetics," *Aquaculture*, vol. 255, no. 1-4, pp. 1–29, 2006.

[59] P. M. Schneider, K. Bender, W. R. Mayr, W. Parson, B. Hoste, R. Decorte, J. Cordonnier, D. Vanek, N. Morling, M. Karjalainen, *et al.*, "Str analysis of artificially degraded dna—results of a collaborative european exercise," *Forensic science international*, vol. 139, no. 2-3, pp. 123–134, 2004.

[60] J. Burger, S. Hummel, B. Herrmann, and W. Henke, "Dna preservation: a microsatellite-dna study on ancient skeletal remains," *ELECTROPHORESIS: An International Journal*, vol. 20, no. 8, pp. 1722–1728, 1999.

[61] P. Craig R, M. T. Koskinen, and J. Piironen, "The one that did not get away: individual assignment using microsatellite data detects a case of fishing competition fraud," *Proceedings of the Royal Society of London. Series B: Biological Sciences*, vol. 267, no. 1453, pp. 1699–1704, 2000.

[62] B. K. Hall, *Keywords and concepts in evolutionary developmental biology.* Discovery Publishing House, 2007.

[63] R. Zardoya, D. M. Vollmer, C. Craddock, J. T. Streelman, S. Karl, and A. Meyer, "Evolutionary conservation of microsatellite flanking regions and their use in resolving the phylogeny of cichlid fishes (pisces: Perciformes)," *Proceedings of the Royal Society of London. Series B: Biological Sciences*, vol. 263, no. 1376, pp. 1589–1598, 1996.

[64] T. Wirth and L. Bernatchez, "Genetic evidence against panmixia in the european eel," *Nature*, vol. 409, no. 6823, p. 1037, 2001.

[65] P. R. Aldrich, J. L. Hamrick, P. Chavarriaga, and G. Kochert, "Microsatellite analysis of demographic genetic structure in fragmented populations of the tropical tree symphonia globulifera," *Molecular ecology*, vol. 7, no. 8, pp. 933–944, 1998.

[66] D. B. Goldstein, G. W. Roemer, D. A. Smith, D. E. Reich, A. Bergman, and R. K. Wayne, "The use of microsatellite variation to infer population structure and demographic history in a natural model system," *Genetics*, vol. 151, no. 2, pp. 797–801, 1999.

[67] R. Frankham, "Genetics and conservation biology," *Comptes Rendus Biologies*, vol. 326, pp. 22–29, 2003.

[68] M. P. Miller, B. J. Knaus, T. D. Mullins, and S. M. Haig, "Ssr_pipeline: A bioinformatic infrastructure for identifying microsatellites from paired-end illumina high-throughput dna sequencing data," *Journal of Heredity*, vol. 104, no. 6, pp. 881–885, 2013.

[69] X. Wang and L. Wang, "Gmata: an integrated software package for genome-scale ssr mining, marker development and viewing," *Frontiers in plant science*, vol. 7, p. 1350, 2016.

[70] R. S. and S. H., "Primer3 on the WWW for General Users and for Biologist Programmers," *Bioinformatics Methods and Protocols*, vol. 132, pp. 365–368, 2000.

[71] S. Metz, J. M. Cabrera, E. Rueda, F. Giri, and P. Amavet, "Fullssr: microsatellite finder and primer designer," *Advances in bioinformatics*, vol. 2016, 2016.

[72] A. J. Robinson, C. G. Love, J. Batley, G. Barker, and D. Edwards, "Simple sequence repeat marker loci discovery using ssr primer," *Bioinformatics*, vol. 20, no. 9, pp. 1475–1476, 2004.

[73] "Sputnik." http://abajian.net/sputnik/. Accessed: 2010-09-30.

[74] L. C. Da Maia, D. A. Palmieri, V. Q. De Souza, M. M. Kopp, F. I. F. de Carvalho, and A. Costa de Oliveira, "Ssr locator: tool for simple sequence repeat discovery integrated with primer design and pcr simulation," *International journal of plant genomics*, vol. 2008, 2008.

[75] T. T., M. W., V. RK, and G. A., "Exploiting EST databases for the development and characterization of gene-derived SSR-markers in barley (Hordeum vulgare L.).," *Theor Appl Genet*, vol. 106, no. 3, pp. 411–422, 2003.

[76] S. Temnykh, G. DeClerck, A. Lukashova, L. Lipovich, S. Cartinhour, and S. McCouch, "Computational and experimental analysis of microsatellites in rice (oryza sativa l.): frequency, length variation, transposon associations, and genetic marker potential," *Genome research*, vol. 11, no. 8, pp. 1441–1452, 2001.

[77] J. W. Davey and M. L. Blaxter, "Radseq: next-generation population genetics," *Briefings in functional genomics*, vol. 9, no. 5-6, pp. 416–423, 2010.

[78] L. Garrett, D. Dawson, G. Horsburgh, and S. Reynolds, "A multiplex marker set for microsatellite typing and sexing of sooty terns Onychoprion fuscatus," *BMC research notes*, vol. 10, p. 756, 12 2017.

[79] B. A. M., L. M., and U. B., "Trimmomatic: A flexible trimmer for Illumina Sequence Data.," *Bioinformatics*, vol. 30, no. 15, pp. 2114–2120, 2014.

[80] M. MARTIN, "Cutadapt removes adapter sequences from high-throughput sequencing reads," *EMBnet.journal*, vol. 17, no. 1, pp. 10–12, 2011.

[81] R. Li, Y. Li, K. Kristiansen, and J. Wang, "SOAP: short oligonucleotide alignment program," *Bioinformatics*, vol. 24, pp. 713–714, 01 2008.

[82] S. L. Magoc, Tanja ;Salzberg, "FLASH: fast length adjustment of short reads to improve genome assemblies," *Bioinformatics*, vol. 27, no. 21, pp. 2957–2963, 2011.

[83] L. Fu, B. Niu, Z. Zhu, S. Wu, and W. Li, "CD-HIT: accelerated for clustering the next-generation sequencing data," *Bioinformatics*, vol. 28, no. 23, pp. 3150–3152, 2012.

[84] W. li and A. Godzik, "Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences," *Bioinformatics*, vol. 22, no. 13, pp. 1658–1659, 2006.

[85] C. Dieffenbach, T. Lowe, and G. Dveksler, "General concepts for PCR primer design," *PCR methods and applications*, vol. 3, pp. 30–37, 12 1993.