Bioinformatics
doi.10.1093/bioinformatics/xxxxxx
Advance Access Publication Date: 2 April 2015
Applications Note

OXFORD

Structural Bioinformatics

Micro-Primers: Pipeline for automation of SSR's screening for population genomics

Filipe Alves [1, 2]

[1] Biology Department, Faculty of Sciences, Porto, 4169-007, Portugal and
[2] Computer Science Department, Faculty of Sciences, Porto, 4169-007, Portugal.

Abstract

Summary: Studies of variation intra/inter population have become an important way to characterize the biodiversity of a species and a powerful tool for conservation programs, since high levels of inbreeding could lead into a whole population extintion in some generations. In this project we exploit the rise of new and cheap sequencing technologies than produce huge amounts of data with the automatic computing to study and characterize individuals and even entire populations through the usage of Microsatellites (SSR), thanks to the new pipeline Micro-Primers. Micro-Primers is a Python and PERL coded pipeline to identify and design PCR primers for amplification of SSR loci. The pipeline takes as input just a fastq file containing sequences from NGS (next generation sequencing) technologies and returns a text file with the primers and their respective melting temperature to easily perform an amplification of the same loci in other individuals of the same species.
Availability: The pipeline is avaiable at the github repository FilAlves/micro-primers
Contact: up201305072@fc.up.pt

1 Introduction

Simple sequence repeats (SSR's) or microsatellites are repeats of DNA motifs that make up genomic repetitive regions. They're evenly distributed throughout the genome and, in conjunction with other Simple Sequence Length Polymorphisms (SSLP), they're are evolutionarily important since they have bigger mutation rates than other DNA regions, making them excellent markers for genotype identification, analysis of genetic diversity and phenotype mapping (Vieira *et al*, 2016).
Nowadays, there is no standard method for doing SSR screening. The process involves just sequencing and checking variations of distribution along the species by hand. Additionally, this process has a fundamental limitation, the high number of different programs needed to perform the SSR analysis, where each program has its own communication protocol. Most of the times the output of one of this programs is not similar to the input the next program needs, so again manual interaction is essential to continue the analysis what could derive in errors due to human manipulation.
The goal of this proposal is to design and implement a novel tool for doing automatically 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). The tool is fed with sequences coming directly from sequencing and is able to detect the SSRs presents in the individual/population and design the primers to reproduce the analysis in a different dataset from the same species.
All individuals of a given species possess the same SSR at the same position. What differs between individuals is the number of repeats of each motif. The evolution of this repeats can be followed in the genealogy and enable the construction of evolutionary histories for each species. The number of different repeats each individual has within the same population is so high that, with few number of this loci, we can almost identify single individuals with a quick and swift test, just like paternity tests, with minimal error rates.

2 Background

Currently it can take up months for laboratories and researches to analyse and select SSR primers for their projects. It requires to run MISA several times without any kind of pre-processing and to select by hand individual candidates for primer3. With no clusterisation or unknowing the number of available loci, most amplifications are inefficient and increasing funds are wasted on reagents. The creation of an automatic pipeline that can speed up the process imune to manual selection bias is of interest for the scientific community.

Currently, all the pipeline of programs to be executed and the manual steps for getting the final results already exist, thus the main objective is to fit all of them into a unique and easy framework that takes the input and generates the final result.

With this is mind several tools and scripts were integrated into one program, capable of discovering SSR's and within the input data and the respective primers for amplification. The tools included in this pipeline are: (i) Trimmomatic (Bolger *et al*, 2014), for sequencer adapters removal. (ii) Cutadapt (Martin *et al*, 2011), for Technology specific adapter removal. (iii) FLASH (Magoc *et al*, 2011), for joining both R1 and R2 sequences, (iv) MISA (Beier *et al*, 2017), for SSR search in the selected sequences, (v) CD-HIT (Weizhong *et al*, 2012), for redundacy removal, (vi) Primer3 (Rozen *et al*, 2000), for primer design. In additions to this tools, PERL scripts coded by Antonio Munoz Merida (https://cibio.up.pt/people/details/amunoz) are included for data output processing.
Python is a popular high-level programming language that supports object-oriented and structural programming. Its focus on easy code readability, easy portability between different operating system, the availability of vast amounts of libraries and automatically memory management as made Python one of the most used programming language in various areas, including scientific applications. Figure 1 shows the complete pipeline.

3 Our Aproach

In this section we describe the two implementations of the Micro-Primers software[1].

3.1 First Implementation

The input file, a Fastq formatted file including multiple sequences, are parsed to Trimmommatic and Cutadapt for adapter removal. The output is then parced to Flash to join both R1 and R2 fragments. Next, only the sequences that start and end with the restriction enzyme pattern are selected. After, for later easier selections, is added a unique id to each sequence. The remaining sequences then are fed to MISA for SSR identification. The length from the end of the SSR to the absolute end of the sequence is also measure and added to MISA output.
The resulting output matrix contains for each sequence (i) id, (ii) number of SSR's present, (iii) SSR type, (iv) SSR size, (v) start position, (vi) end position and (vii) length to end.
It is necessary to have certain length from both sides of the microsatelite to the end of the sequence where the restriction enzyme cut since the final objective is to design primers to amplify the SSR itself. Amplification primers usually have a length around 20-25 nucleotides and it is known that the complementarity in the edges of a double-strand sequence is less strong that inside the borders. So the primer to amplify the interest area should be in this case at a distance from the border of 50.
With this in mind, the first filter in the pipeline involves the manual selection of candidate sequences that accomplish the criteria specified by the user. In this testing stage only sequences that (i) are not compost (c) or 1 letter sequence (p1) SSR's, (ii) start position and length to absolute end form SSR's end position above or equal to 50 are selected.
In the next step is extracted the SSR sequence form the fragments for alignment of the flanking regions and the removal of redundancy, via CD-HIT. Subsequently it's assigned a cluster to each SSR, representing the similarity between the different SSR's present in each sequence. The information is added to the pre-existing SSR matrix.
A new manual selection happens, in which the options are also user specified. Only one SSR per cluster with more than 5 repetitions is chosen, at random.
An allele is every single variation of a same region. In the microsatellites they are any different number of repeats of the same pattern in the tandem sequence. Thus, the higher the number of alleles we have for a SSR, the less number of SSRs we need to characterize a population or individual.
A new file is created with only the ID, pattern, start and end positions.
The resulting file contains the sequences ready to be parced to Primer3. In this first implementation the default `CTM_Setting_Long` setting is used. The resulting file is then filtered by a script that only choses the best primers that follow laboratory criteria and converts it in a more readable text file. At this stage the laboratory criteria were ignored.
All intermediate files were saved in a temporary folder, which is erased at the end of the pipeline.

3.2 Second Implementation

In this second implementation the manual steps in the first implementation were converted in a python coded script. In addition, a new file "settings.txt" file, including the default values mentioned above, was created were the user can specify the conditions for the 1st and 2nd filters. Since each company/Bioinformatician uses a specific primer3 setting file, it's location can also be specified in the settings file.
For the first filter is used the function csv_picker.py (Appendix 1). It splits each line of the MISA output by tab separation and, knowing the index of the SSR type and SSR length columns, selects only the lines that follow the user criteria. After, it writes all the approved lines in a new document, containing only the ID, SSR type, start and end positions.
For the second filter, the function selected_micros (Appendix 1) creates a dictionary in which it uses as keys the cluster type and as values the ID of one sequence. While itinerating for each line tab seperated line in the file, if a cluster type still doesn't contain a dictionary entry, a new entry is created using the Sequence ID of the given line. Given the nature of cluster creation only a single representative of each cluster is necessary. A new file is created containing only the ID sequences, ready to be parsed to SSR Extraction Script.

4 Experimental Results

The pipeline requeries that the tools CutAdapt, CD-HIT, and FLASH to be installed. At this point the pipeline has optimal perfomance using Unix based and MAC-OS systems but it's not compatible with Windows. The maximum input size is also unknown.
At this early stage is difficult to assess the pipeline performance since no time was available to use different samples and since no similar pipeline is availed for public use, no performance comparison can be done. For a basic benchmark, using two 225,8 Mb Fastq containing R1 and R2 sequences, using an 8 Gb RAM DDR3 and a Quad-Processor at 2.50 Ghz powered computer,the pipeline took 1 minute and 47 seconds to output a result.

[1] Appendix 1 shows the Micro-Primers' code.

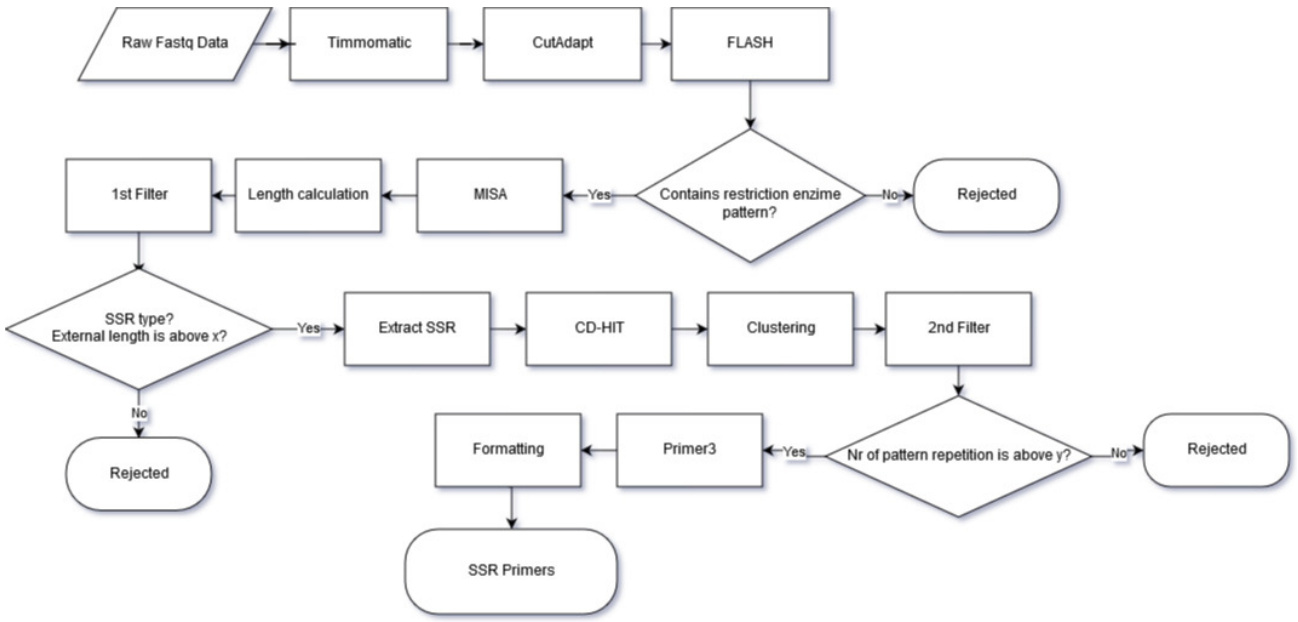

Figure 1: Overview of Micro-primers pipeline.

5 Conclusion and Future Work

The use of micro-primers provides an unprecedented availability of candidate SSR's at a much more realiable and fast pace than before. This abundance permits selection of markers that may be most suitable for specific applications or particular organisms.

This being only an initial implementation there's a lot of space for improvement. Converting all PERL script into a single python script, pipeline parallelization, and removal of temporary files are only a few points were performance can be improved. Also, regarding data quality, more options can be added to amplify the usefulness and its malleability to different samples. A graphic user interface (GUI) and Windows compatibility can also be added to facilitate user configuration.

Concluding, is easy to perceive the enormous potential of micro-primers for the creation the of a framework that can be easily used by researchers in several host operation systems without the assistance of any data analyst and, in the following months, it's expected the addition of more options for the user and an increase of efficency.

References

Vieira, M. L. C., Santini, L., Diniz, A. L., Munhoz, C. de F. (2016). Microsatellite markers: what they mean and why they are so useful. *Genetics and Molecular Biology*, doi:10.1590/1678-4685-GMB-2016-0027.

MARTIN, Marcel (2011). Cutadapt removes adapter sequences from high-throughput sequencing reads. *EMBnet.journal*, **99**, 10-12. doi:10.14806/ej.17.1.200.

Weizhong *et al*, (2012). CD-HIT: accelerated for clustering the next generation sequencing data. *Bioinformatics*, **28**, 3150-3152. doi: 10.1093/bioinformatics/bts565

Magoc, Tanja ;Salzberg, Steven L. (2011). FLASH: fast length adjustment of short reads to improve genome assemblies. *Bioinformatics*, **27**, 2957-2963. doi:10.1093/bioinformatics/btr507

Rozen S., Skaletsky H. (2000). Primer3 on the WWW for General Users and for Biologist Programmers. *Bioinformatics Methods and Protocols*, **132**. doi:10.1385/1-59259-192-2:365

Bolger, A. M., Lohse, M., Usadel, B. (2014). Trimmomatic: A flexible trimmer for Illumina Sequence Data. *Bioinformatics*, **30** 2114-2120. doi: 10.1093/bioinformatics/btu170

Beier, Sebastian *et al*. (2017). MISA-web: a web server for microsatellite prediction. *Bioinformatics*, **33** 2583-2585. doi: 10.1093/bioinformatics/btx198

1 Micro-Primers' Code

```
#Pipeline
import os
from software.scripts import picker, config

#Create empty file for importing python scripts
os.system("touch software/scripts/__init__.py")

#Reading settings
settings = config.config("config.txt")
#Creation of hidden temp file
if os.path.isdir(".temp/") == False:
    os.system("mkdir .temp")
if os.path.isdir("logs/") == False:
    os.system("mkdir logs")

#Sequences Triming of Sequencer adapters
def trimmomatic(R1, R2):
    os.system("echo 'Trimmomatic working...'")
    os.system(
        "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:/home/filalves/projecto/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)
        )

#Adapters removal (specifics from technology)
def cutadapt(a, g):
    os.system("echo 'Cutadapt working...'")
    os.system("cutadapt -a %s -g %s -o .temp/cut_out_nolink_R1.fastq .temp/trim_out_trimmed_R1.fastq > logs/cut_log_r1.txt" %
    os.system("cutadapt -a %s -g %s -o .temp/cut_out_nolink_R2.fastq .temp/trim_out_trimmed_R2.fastq > logs/cut_log_r2.txt" %

# Fusion of R1 and R2 files
def flash():
    os.system("echo 'Flash working...'")
    os.system("software/FLASH-1.2.11/flash .temp/cut_out_nolink_R1.fastq .temp/cut_out_nolink_R2.fastq -M 220 -o .temp/flash_
              " tee logs/flash.log > logs/flash_log.txt")

# Selecion of fragments that start and ends with the pattern of the restriction enzyme
def grep():
    os.system("echo 'Selecting sequences with restriction enzime patterns...'")
    os.system("grep -B1 '^GATC\(\w*\)GATC$' .temp/flash_out.extendedFrags.fastq | sed 's/^@/>/' | perl -pe 's/--\n//g' > .tem

#Change id's
def ids():
    os.system("echo 'Changing ids...'")
    os.system("perl software/scripts/changeids.pl .temp/grep_out.fasta")

#Search Microssatelites
def misa():
    os.system("echo 'Misa working...'")
    os.system("perl software/scripts/misa.pl .temp/ids_out.fasta 2> logs/misa_log.txt")

#Length Calculation for later selection of valid microsatellites
def length_calc():
    os.system("echo 'Calculating sequences lengths...'")
    os.system("perl software/scripts/extraelength.pl .temp/ids_out.fasta")
```

```
#Adds length to end of the sequences to misa output
def length_add():
    os.system("echo 'Adding length to misa output...'")
    picker.length_merger(".temp/misa_out.misa", ".temp/length_calc_out.fasta", ".temp/length_add_out.misa")


#Selection of microssatelites with enough space for primer
def good_micros(dist, rep, exclude):
    os.system("echo 'Selecting good microsatellites...'")
    picker.csv_picker(".temp/length_add_out.misa", ".temp/good_micros_out.fasta",
                ".temp/good_micros_table_out.misa",dist, rep, exclude)

#Extraction of the microsatellite sequence from allignement of fragments with flanking regions
def splitSSR():
    os.system("echo 'splitSSR working...'")
    os.system("perl software/scripts/splitSSR.pl .temp/ids_out.fasta .temp/good_micros_out.fasta")

#Removal of Redundacy
def cdhit():
    os.system("echo 'CD-HIT working...'")
    os.system("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

#Cluster assignement
def cluster():
    os.system("echo 'Calculating number of sequences for each cluster...'")
    os.system("perl software/scripts/asign_cluster.pl .temp/cdhit_out.txt.clstr")

#Adding cluster information to Microssatelites table
def cluster_info():
    os.system("echo 'Adding information to the table of microsatellites...'")
    os.system("perl software/scripts/attach_cluster_info.pl .temp/good_micros_table_out.misa .temp/clusters_out.txt")

# Selecting one sequence per cluster
def selected_micros():
    os.system("echo 'Selecting one sequence per cluster...'")
    picker.selected_micros(".temp/cluster_info_out.txt", ".temp/selected_micros_seqs.txt", ".temp/selected_micros_tabs.txt")

#Creating input file for Primer3
def create_pseudofasta():
    os.system("echo 'Creating Primer3 input file...'")
    os.system("perl software/scripts/extraeseqs.pl .temp/ids_out.fasta .temp/selected_micros_seqs.txt")

#Primer design and creation
def primer3():
    os.system("echo 'Creating Primers...'")
    os.system("software/primer3/src/./primer3_core -default_version=2 -p3_settings_file=CTM_settings_long.txt "
              ".temp/pseudo_out.fasta -output=.temp/micros_selected_long.primers")

#Selection of primers following laboratory criteria
def select():
    os.system("echo 'Selecting best primers...'")
    os.system("perl software/scripts/select_oligos.pl .temp/micros_selected_long.primers .temp/selected_micros_tabs.txt > log

#Removal of .temp directory
def junk():
    os.system("rm -r .temp/")

#Pipeline
trimmomatic(settings[0], settings[1])
cutadapt(settings[2], settings[3])
flash()
grep()
```

```
ids()
misa()
length_calc()
length_add()
good_micros(int(settings[4]), int(settings[5]), settings[6])
splitSSR()
cdhit()
cluster()
cluster_info()
selected_micros()
create_pseudofasta()
primer3()
select()
junk()
os.system("echo 'Done!'")
```

2 Picker' Code (First and Second Filter)

```
def selected_micros(rf, of_sel_micros, of_id_micros):
    readfile = open(rf, "r")
    outfile1 = open(of_sel_micros, "w")
    outfile2 = open(of_id_micros, "w")
    dic_cluster = {}
    for line in readfile:
        #Split by tab
        selected_line = line.split("\t")
        #Creating dictionary for selection of one sequencing per cluster
        if not selected_line[9] in dic_cluster.keys():
            dic_cluster[selected_line[9]] = selected_line[0]
            #Creating outfile with sequenceID, ssr, start and end positions of the ssr
            selected_micros = list( selected_line[i] for i in [0, 3, 5, 6])
            outfile2.write("\t".join(selected_micros) + "\n")
    #creating tab_selected
    for keys,values in dic_cluster.items():
        outfile1.write(values + "\n")

def csv_picker(rf, of_micros_good, of_micros_tab, dist, rep, exclude):
    readfile = open(rf, "r")
    outfile = open(of_micros_good, "w")
    outfile2 = open(of_micros_tab, "w")
    #Minimal number of bases after and before SSR
    min_ext_dist = dist
    #Mnimal repetitions of SSR
    min_rep = rep
    #Types of ssr to exclude from further search
    exclude_ssr = exclude
    for line in readfile:
        #Split by tab
        selected_line = line.split("\t")
        #Select line which do not contain c, c* and p1 type SSR
        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])
```

```
                outfile.write("\t".join(good_micros))
                outfile2.write("\t".join(selected_line))
    #Adding "\n" to the end of the file. It alsos messes with splitSSR script
    outfile.write("\n")

def length_merger(rf_csv, rf_length, of):
    readfile_csv = open(rf_csv, "r")
    readfile_length = open(rf_length, "r")
    outfile = open(of,"w")
    #Creating dicitonary with SeqID and ssr length
    dic_length = {}
    # Dictionary creations
    for line in readfile_length:
        #Spliting file by tabs
        selected_line = line.split("\t")
        #Saving only 10 first caracters of ID.
        dic_length[selected_line[0][0:10]] = selected_line[1]
    for line in readfile_csv:
        selected_line = line.split("\t")
        #Removing \n from end tab
        selected_line[6] = selected_line[6].rstrip()
        #Creating new tab with ssr length
        if selected_line[0][0:10] in dic_length.keys():
            selected_line.append(dic_length[selected_line[0][0:10]])
            outfile.write("\t".join(selected_line))
```

3 Config' Code (Config reader)

```
def config(rd):
readfile = open(rd, "r")
#Saves settings parameters as list
settings = []
for line in readfile:
    #Split line by "="
    selected_line = line.split("=")
    #Selects only the values after "="
    if len(selected_line) > 1:
        selected_line[1] = selected_line[1].rstrip()
        settings.append(selected_line[1])
#Converts ssr_type from string to list
ssr_type = settings[6].split(",")
settings[6] = ssr_type
return settings
```