Verbetering van de “effector identification pipeline”

Extern project aan de Universiteit van Amsterdam (UvA)

Aldo Vree (s1109403)

Aron van Beelen (s1082644)

Lieke Vree (s1108210)

Timo de Graaf (s1105224)

Bin3

BPEXA

1e gelegenheid

Jeroen Pijpe (projectbegeleider)

Like Fokkens (extern projectbegeleider) 26 januari 2019

Inhoudsopgave

Samenvatting …………………………………………………………………………………… 3

Inleiding …………………………………………………………………………………………. 4

* Projectomschrijving …………………………………………………………………… 4
* Achtergrondinformatie ………………………………………………………………... 4
* Doelstellingen …………………………………………………………………………. 6

Materiaal en methoden ……………………………………………………………………….. 7

* MimpFinder script …………………………………………………………………….. 8
* EffectorFinder script ………………………………………………………………….. 9
* Augustus script ……………………………………………………………………….. 10
* Presence\_absence script ……………………………………………………………. 11
* Clustering script ………………………………………………………………………. 12
* 04.cluster\_and\_plot\_heatmap3.R script …………………………………………… 13

Resultaten ……………………………………………………………………………………… 14

* Flowchart oude en nieuwe pipeline ………………………………………………… 14
* Nieuwe pipeline en extra bestanden .………………………………………………. 15
* Belangrijkste verschillen ……………………………………………………………… 19
* Performance test ……………………………………………………………………… 21

Discussie ……………………………………………………………………………………….. 22

* Parallellisatie, modularisatie en documentatie …………………………………….. 22
* Performance test ……………………………………………………………………… 22
* Nieuwe versies van gebruikte tools …………………………………………………. 22
* Input genoom ………………………………………………………………………….. 22
* Oude pipeline met testdata …………………………………………………………... 23
* Heatmaps ……………………………………………………………………..……….. 23
* Toekomst ……………………………………………………………………………… 23

Conclusie ……………………………………………………………………………………….. 23

Referentielijst …………………………………………………………………………………… 24

Bijlagen ………..………………………………………………………………………………… 25

* Bijlage 1 – 01a.mimpfinder\_combine\_to\_putefflist\_MetStop.py …………………. 25
* Bijlage 2 – 01b.mimpfinder\_combine\_to\_putefflist\_AUGUSTUS.py …………….. 34
* Bijlage 3 – 02.cluster\_putefflists.py ………………………………………………….. 46
* Bijlage 4 – 03.local\_blast\_clustered\_putefflist\_to\_pres-abs\_table.py …………… 49
* Bijlage 5 – 04.cluster\_and\_plot\_heatmap3.R ………………………………………. 54
* Bijlage 6 – FoEC.py …………………………………………………………………… 57
* Bijlage 7 – MimpFinder.py ……………………………………………………………. 63
* Bijlage 8 – MetStop.py ………………………………………………………………… 66
* Bijlage 9 – augustus.py ……………………………………………………………….. 71
* Bijlage 10 – presence\_absence.py ………………………………………………….. 79
* Bijlage 11 – clustering.py ……………………………………………………………... 82
* Bijlage 12 – INSTALL.sh ……………………………………………………………… 85
* Bijlage 13 – config.ini ………………………………………………………………….. 86
* Bijlage 14 – Snakefile …………………………………………………………………. 87
* Bijlage 15 – README.txt ……………………………………………………………... 90

# Samenvatting

Het *Fusarium oxysporum* soorten complex is een groep schimmels die gezamenlijk verantwoordelijk zijn voor ziektes in meer dan 100 economisch belangrijke gewassen.4 Individuele stammen zijn gastheer specifiek. Om een gastheer te kunnen koloniseren scheiden schimmels kleine eiwitten, genaamd effectoren, uit. Die effectoren kunnen bijvoorbeeld het immuunsysteem van de gastheer hinderen. Omdat effectoren nauw betrokken zijn bij de interactie tussen de schimmel en zijn gastheer, kunnen we verwachten dat deze gastheer-specifiek zijn en dus gebruikt kunnen worden om de gastheervoorkeur van een stam te voorspellen. Het identificeren van effectoren is dus erg belangrijk.

Uit onderzoek blijkt dat de effector-genen op het genoom geassocieerd worden met transposons, waaronder MIMPs. Deze associatie vormt de kern van een pipeline die is geschreven om nieuwe effectoren in *Fusarium oxysporum* genomen te identificeren. Na het identificeren van nieuwe effectoren, worden deze geclusterd om te bepalen welke effectoren gastheer-specifiek zijn en of gastheervoorkeur op basis van het ‘effector repertoire’ voorspeld kan worden.

De huidige pipeline is echter erg onoverzichtelijk en daarom moeilijk uit te breiden of aan te passen: de pipeline is niet modulair opgezet en de functies zijn niet goed gedocumenteerd. Het is lastig om de pipeline met verschillende parameters te runnen. Bovendien zijn zowel de taal waarin de pipeline is geschreven (Python2) als de tools verouderd. Tenslotte is de pipeline moeilijk te parallelliseren. De doelstelling van het project was het verbeteren van de huidige pipeline zodat het (i) makkelijker aan te passen en te gebruiken is, (ii) zowel de pipeline zelf als de tools die worden gebruikt te updaten naar de nieuwste versies en (iii) de mogelijkheid tot parallellisatie bieden.

De verschillende onderdelen van de oude pipeline zijn geherstructureerd en herschreven in nieuwe scripts: MimpFinder.py, MetStop.py, augustus.py, presence\_absence.py en clustering.py. Deze nieuwe code is grotendeels object-georiënteerd geschreven. Er is met name op gelet dat er korte functies geschreven zijn en alle functies een uitgebreide beschrijving hebben, waardoor de pipeline overzichtelijker en makkelijker aan te passen is. Er zijn een install file, config file en README toegevoegd zodat de pipeline makkelijker te gebruiken is. Voor het verbeteren van de pipeline is gebruik gemaakt van nieuwere software, zoals Python 3.6 of hoger, Augustus 3.3.3 en SignalP 5.0. De pipeline zelf, die de verschillende onderdelen aanroept, is geschreven in snakemake om parallellisatie mogelijk te maken. Uit de performance test die is uitgevoerd blijkt dat de nieuwe pipeline (met parallellisatie) maar liefst 34.4% sneller is. Dit is belangrijk omdat de pipeline vaak op grote datasets wordt toegepast.

De pipeline is zeker verbeterd. De code van de pipeline is modulairder en leesbaarder, de documentatie is verbeterd en de pipeline is gebruiksvriendelijker door het bijvoorbeeld toevoegen van een install file en config file. Daarnaast is de nieuwe pipeline sneller dan de oude en is parallellisatie mogelijk. De pipeline is echter nog niet helemaal bug vrij, dus hier moet in de toekomst nog aan gewerkt worden. Ook de README moet in de toekomst verbeterd worden, er moet namelijk nog een uitgebreide projectomschrijving en doel van de pipeline geschreven worden.

# Inleiding

## Projectomschrijving

De Molecular Plant Pathology groep van de Universiteit van Amsterdam heeft een effector-voorspellings pipeline ontwikkeld die effectoren, in de plantpathogene schimmel *Fusarium oxysporum*, identificeert. De pipeline wordt veel gebruikt in de wetenschappelijke gemeenschap en in de industrie.

De pipeline is functioneel, maar het heeft verbetering nodig. Om dit te bereiken hebben wij het volgende gedaan:

1. De scripts en functies gereorganiseerd, zodat de code een gemoduleerde opbouw heeft. Hierdoor kunnen nieuwe extensies en functies makkelijker geïmplementeerd worden.
2. De pipeline gestructureerd en aangeroepen in Snakemake1. Dit maakt parallellisatie mogelijk; data worden parallel verwerkt in plaats van opeenvolgend. Ook maakt Snakemake de pipeline reproduceerbaar en schaalbaar.
3. De scripts en functies in de scripts goed gedocumenteerd: essentieel voor de gebruiksvriendelijkheid en leesbaarheid van de code.
4. Nieuwe en/of verbeterde extensies en functies toegevoegd aan de pipeline.

## Achtergrondinformatie

Het *Fusarium oxysporum* (FO) soorten complex (FOSC) is een groep die bestaat uit plantpathogene schimmels. Het FOSC in zijn geheel heeft een zeer groot gastheerbereik, maar individuele stammen infecteren een of een paar gastheren; deze stammen zijn gegroepeerd in forma specialis (ff.spp.); op basis van bekende gastheervoorkeur. Het FOSC is verantwoordelijk voor ziektes in 120 plantensoorten4, waaronder economisch belangrijke gewassen zoals: *Cucumis melo*; suikermeloen, *Cucumis sativu*; komkommer en *Solanum lycopersicum*; tomaat.2

Ondanks dat het bekend is dat *formae speciales* een groot gastheerbereik heeft, is het precies bepalen welke gastheren de stammen infecteren op basis van fylogenie niet betrouwbaar. Dit komt door horizontale genoverdracht3: een biologisch proces waarbij genen tussen niet-familiaire organismen worden overgedragen. Horizontale genoverdracht levert een probleem2; de stammen in forma specialiskunnen niet betrouwbaar geïdentificeerd worden met generieke identificatiemethoden. Met een alternatieve methode, ‘disease assays’, kunnen gastheer en (schimmel)soorten onderscheiden worden. Helaas zijn disease assays tijdrovend en arbeidsintensief, dus een moleculaire screening op basis van *formae-speciales-*specifieke DNA-sequenties zou ideaal zijn. Echter, kennis over de genetische basis van gastheerspecificiteit is gelimiteerd.

Om gastheren te infecteren, gebruikt FOeffectoren. Effectoren2 zijn eiwitten die uitgescheiden worden en vermoedelijk het immuunsysteem van de gastheer beïnvloeden. Aan de hand van de set van effectoren van een stam, kunnen we diens gastheervoorkeur, dus bij welke forma specialis de stam hoort.

Biologisch onderzoek2 naar tomaat-infecterende *F. oxysporum* f.sp *lycopersici* (Fol), heeft aangetoont dat chromosoom 14 allen behalve één “Secreted In Xylem”-genen (SIX) bevat. Deze genen zijn vermoedelijk de effector-genen. Uit onderzoek naar deze genen blijkt dat zij geassocieerd worden met twee soorten transposons5, namelijk:

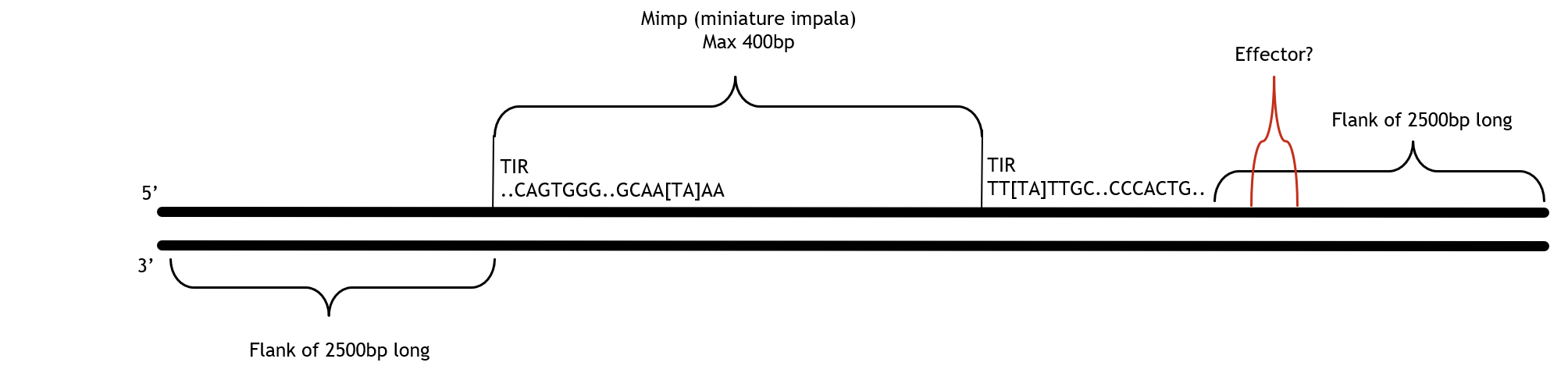
1. “miniature impala” (MIMP);

Een transposon, gevonden upstream van het DNA.

1. mFot5.

Een transposon, gevonden in het Open Reading Frame (ORF) downstream van het DNA.

Beide transposons vallen onder de klasse van de “Miniature Inverted-repeat” Transposable Elements (MITEs). MITEs bevatten “terminal inverted repeats” (TIRs) van 27-30 nucleotiden. TIRs zijn herhaalde nucleotiden die omgekeerd aan de uiteinden van een DNA-element zittten.2 In figuur 1 in een schematische weergave van de MIMP met de bijbehorende TIRs te zien.



**Figuur 1.** *In de Figuur is de miniature impala (MIMP) met zijn bijbehorende TIRs te zien. Ook zijn de upstream en de downstream flank te zien waarop gezocht wordt naar effectoren.*

Dus, door het verband tussen MITEs en effector-genen in FO-genomen te bepalen kan het repetoire aan effectoren achterhaald worden. Het effectoroom van verschillende FO-genomen wordt vergeleken om zo te besluiten welk repertoire aan effectoren gelijk zijn van stammen binnen *formae speciales.* Ook kan van de vermoedelijke effector-genen bepaald worden, met behulp van de patronen in de “presence/absence”-test met het effectoroom data, welk gastheerbereik zij hebben zonder disease assays toe te passen.

Het bepalen van het effectoroom van de FO-genomen wordt gedaan met een effector-voorspellings pipeline. De bestaande pipeline bestaat uit verschillende scripts die aangeroepen worden in het FoEC.py script (bijlage 6), namelijk:

* 01a.mimpfinder\_combine\_to\_putefflist\_MetStop.py (bijlage 1);

Het script zoekt MIMPs in het genoom, vervolgens worden ORF’s in 2500 basenparen downstream van de MIMP TIR gezocht, daarna wordt SignalP-4.1 gebruikt om signaalpeptidesequenties te vinden in de ORF’s met een cut-off van D=0,550. Ten slotte worden de putatieve effectoren weggeschreven in een FASTA-file.

* 01b.mimpfinder\_combine\_to\_putefflist\_AUGUSTUS.py (bijlage 2);

Het script zoekt MIMPs in het genoom, vervolgens worden genen voorspeld downstream van de MIMP TIR, met behulp van AUGUSTUS, daarna wordt SignalP-4.1 gebruikt om signaalpeptidesequenties te vinden, in de voorspelde genen van AUGUSTUS, met een cut-off van D=0,550. Ten slotte worden de effectoren weggeschreven in een FASTA-file.

* 02.cluster\_putefflist.py (bijlage 3);

De FASTA-files met vermoedelijke effectoren worden geclusterd en de langste sequentie per cluster wordt weggeschreven naar één FASTA-file.

* 03.local\_blast\_clustered\_putefflist\_to\_pres-abs\_table.py (bijlage 4);

De FASTA-file met de langste eiwitsequentie per cluster worden geBLAST tegen alle FO-genomen waaruit een Tabel met similarity cores volgt.

* 04.cluster\_and\_plot\_heatmap3.R. (bijlage 5);

De Tabel met de similarity scores wordt omgezet naar een heatmap met de presence-absence per effector in de FO-genomen.

Dus, in grote lijnen zoekt de pipeline naar genen in het genoom en filtert deze genen op basis van signaalpeptidesequentie. Signaalpeptidesequenties zijn typerend voor secretie-eiwitten, dus deze eiwitten kunnen potentiële effectoren zijn. De gefilterde genen met signaalpeptidesequenties worden geclusterd en de langste sequentie van ieder cluster wordt tegen het input genoom geBLAST. Uiteindelijk komen de sets aan effectoren in een Tabel, met een similarity score voor iedere FO-genoom. De Tabel met de scores wordt omgezet naar een heatmap. Ten slotte geeft de heatmap de presence-absence van de effectoren in de FO-genomen weer.

# Doelstellingen

Biologische doelstelling van de pipeline: het vergelijken van het effectoroom in *Fusarium oxysporum*-genomenom te bepalen of het effecetor repertoire de gastheervoorkeur van een stam kan voorspellen, en zo ja, welke combinatie van effectoren gastheervoorkeur bepalen

Doelstelling van het project: het verbeteren van de huidige pipeline. De pipeline wordt gebruiksvriendelijker gemaakt door middel van goede documentatie. Dit wordt gedaan met een config-bestand, heldere en uitgebreide documentatie bij ieder script en haar functies. De pipeline wordt versneld door parallellisatie. De parallellisatie wordt gedaan met Snakemake. De software in de pipeline wordt geüpdatet, zodat de pipeline is voorzien van de laatste versies. De pipeline wordt modulairder gemaakt door de functies te herschrijven zodat zij één specifieke taak uitvoeren. Daarbij worden de scripts gedeeltelijk object-georienteerd geschreven, zodat nieuwe upgrades makkelijker geimplemteerd kunnen worden

# Materiaal & Methoden

Om de pipeline te verbeteren werd gekozen om het te reorganiseren in verschillende modules (zie bijlage voor de code), namelijk:

1. MIMP Finder;
   * MimpFinder.py
2. Effector Finder;
   * MetStop.py
   * Augustus.py
3. Effector Clustering;
   * Clustering.py
4. Presence-Absence;
   * Presence-Absence.py
5. Clustering & Visualisation.
   * Cluster\_and\_plot\_Heatmap3.R

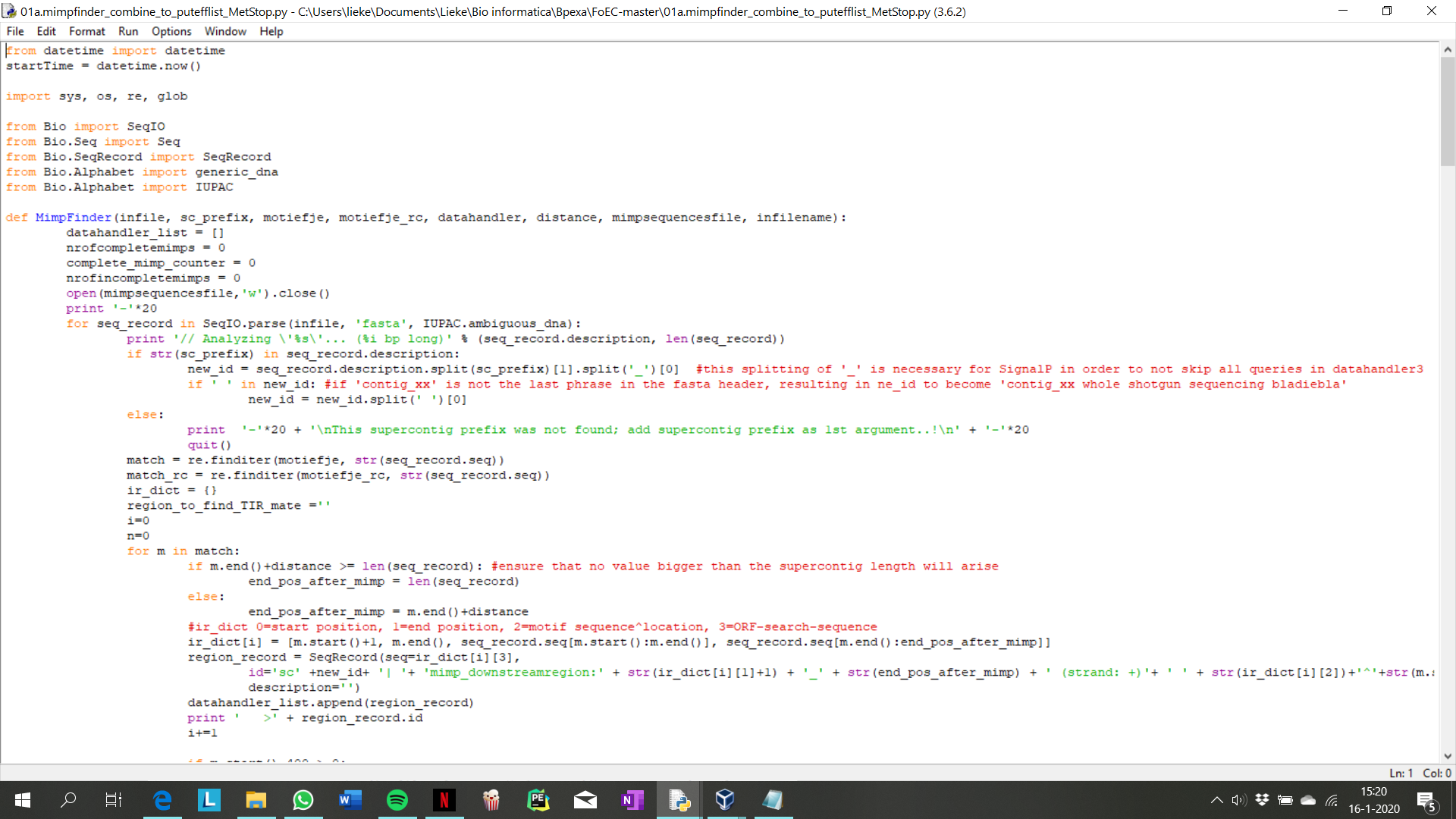
## MimpFinder script:

*Materiaal*

* Python 3.6;
* Biopython;
* PC met Linux-omgeving;
* 01a.mimpfinder\_combine\_to\_putefflist\_MetStop.py script (bijlage 1);
* Snakemake.

*Methoden*

Allereerst werd uitgezocht hoe het bestaande script werkte en hoe de vorige gebruiker het vinden van de MIMPs had aangepakt. Een deel van het bestaande script is te zien in Figuur 2, voor het volledige script zie bijlage 1.



**Figuur 2.** *In de Figuur is het begin van het bestaande script te zien. Het script bevat lange, onoverzichtelijke functies met te lange regels.*

Omdat dit script erg onduidelijk was, werd vervolgens besloten om niet het bestaande script te herschrijven, maar een volledig nieuw script te schrijven, waarbij wel naar het bestaande script gekeken kon worden voor ideeën. In het script werd gebruik gemaakt van functies van Biopython, deze werden geïmporteerd aan het begin van het script, namelijk SeqIO van Bio en IUPAC van Bio.Alphabet. Ook werden de modules ‘re’ en ‘configparser’ geïmporteerd voor gebruik in het script. Voor het testen van het script is gebruik gemaakt van bestaande testdata, bestaande uit fasta files van delen van het genoom van 3 verschillende organismen uit het *Fusarium oxysporum* soorten complex, namelijk Fol4287, Fom001 en Foq001. In het script werd eerst een functie geschreven om de sequentie uit het fasta bestand te halen en deze was in combinatie met de sequentie id toegevoegd aan een lijst. Vervolgens werd de daadwerkelijke MimpFinder functie geschreven (en MimpFinder\_rc voor de reverse complement), welke de MIMPs vond in de sequentie aan de hand van een meegegeven sequentie die werd gedefinieerd in de config file. Ook is er nog een TIRFinder functie geschreven die zoekt naar ‘complete’ MIMPs, dit betekent dat er een tweede MIMP binnen 400 nucleotiden voor of na de eerste mimp zat. Tenslotte was de functie WriteToFile geschreven die de gevonden MIMPs en TIRs wegschrijft naar een file. Deze functie kreeg de output van de MimpFinder of van de TIRfinder en schrijft deze weg naar een fasta file. Bij alle functies is daarnaast gelet op goede documentatie en uitleg wat de functie doet. Hierbij was eerst de functiebeschrijving te zien en vervolgens welke parameters er in de functie gebruikt werden met een beschrijving van wat ze inhielden. De output werd zo gemaakt dat deze hetzelfde was als de output van het originele programma, zodat deze in verdere stappen in het programma goed gebruikt kon worden. Er is uiteindelijk voor gekozen om dit script niet object-georienteerd te schrijven, omdat het bij dit script beter werkt om procesmatig te schrijven.

## EffectorFinder script (MetStop):

*Materiaal*

* Python 3.6;
* PC met Linux-omgeving;
* SignalP-5.0;
* 01a.mimpfinder\_combine\_to\_putefflist\_MetStop.py script (bijlage 1);
* Snakemake.

*Methoden*

Het nieuwe script is geschreven met behulp van het oude script. Hierbij zijn functies die relevant waren voor het nieuwe script gebruikt als inspiratie.

Ten eerste werd het script object-georiënteerd (OO) geschreven met behulp van een klasse genaamd “MetStop”. De klasse heeft één attribuut: de FASTA-file van de MimpFinder module.

Ten tweede werden verschillende methoden gemaakt, genaamd: get\_file; een get-methode die het attribuut teruggeeft, set\_file; een set-methode die het attribuut verandert met variabele n\_file, parser; een methode die de config-file parsed en de inhoud in een instantie maakt, Translate\_Downstream\_Regions; een methode die de downstream regio van 2500 basenparen neemt en de ORF’s ervan bepaald. De ORF’s worden in een lijst toegevoegd en veranderd in een instantie, ORF\_Finder; een methode die de lijst van ORF’s neemt en sequenties, groter dan 26 aminozuren en kleiner dan 300 aminozuren, in een lijst toevoegt. De sequenties worden bepaald met een startcodon en stopcodon, ORF\_Writer; de functie schrijft de lijst met sequenties naar een file, SignalP; een methode die de tool SignalP-5.0 aanroept en signaalpeptidesequenties bepaalt van de sequenties. Alleen signaalpeptidesequenties met een cut-off van D > 0,550 zijn een match. Dus, SignalP-4.0 is vervangen door SignalP-5.0, SignalP\_writer; een methode die de output van SignalP-5.0 wegschrijft naar een file met een relevante header uit de Header\_Maker methode, Header\_Maker; een methode die relevante headers maakt voor iedere sequentie.

Ten derde werden de klasse en methodes zo geschreven dat zij leesbaar zijn; goede documentatie, met behulp van reStructuredText format, per methode en duidelijke namen voor de variabelen zijn toegepast. Ook zijn de methodes zo geschreven dat zij geen redundantie bevatten.

Ten slotte is het nieuwe script in de Snakemake-file verwerkt om parallellisatie mogelijk te maken.

## EffectorFinder script (Augustus):

*Materiaal:*

* Python 3.6;
* PC met Linux omgeving;
* SignalP-5.0;
* Augustus versie 3.0;
* Biopython 3.6
* Bcbio-gff;
* 01b.mimpfinder\_combine\_to\_putefflist\_AUGUSTUS.py (bijlage 2) .

*Methoden:*

Voor het maken van het nieuwe Augustus.py script is eerst gekeken wat de vorige ontwikkelaar heeft gedaan in de oude Augustus script en waarom hij het zo gedaan heeft. Als eerst werd het 01b.mimpfinder\_combine\_to\_putefflist\_AUGUSTUS.py script helemaal begrepen. Vervolgens werd beredeneerd wat de meest praktische en logische manier is om het nieuwe Augustus.py script te maken.

Eerst werd ervoor gezorgd dat het script Augustus kon runnen met de gevonden MIMPs uit de vorige module. De output hiervan werd daarna verwerkt, zodat alleen de genen overblijven met hun eiwitsequentie. Hierop werd SignalP-5.0 gebruikt met de volgende parameters: -fasta, -short -org ‘euk’. De -fasta parameter geeft aan wat voor type file de input is. De -short parameter geeft aan dat de output van SignalP één regel per sequentie moet zijn met de score of er een signaalpeptidesequentie aanwezig was of niet. En als laatst geeft de -org ‘euk’ waarvan de sequenties afkomstig zijn. Door deze parameter mee te geven, gebruikt SignalP netwerken en modellen die getraind zijn op andere eukaryote sequenties. Vervolgens werd de output van SignalP verwerkt zodat alleen sequenties met een signaalpeptidesequentie overblijven.

Uiteindelijk werd de informatie van plek en sequentie van de signaalpeptiden gebruikt om de juiste effectoren te vinden. Met behulp van deze informatie werd de effectoren uit het bestand met de MIMPs geknipt. Daarna werd het weggeschreven naar een file. Als er geen effectoren gevonden werden dan werd “No effectors found….” weggeschreven. Tot slot, alle tijdelijke files werden in het proces verwijderd.

## 

## Presence\_absence script:

*Materiaal:*

* Python 3.6;
* PC met Linux omgeving;
* NCBI-blast+ versie 2.10.0
* 03.local\_blast\_clustered\_putefflist\_to\_pres-abs\_table.py (bijlage 4)

*Methoden:*

Eerst werd gekeken naar het script 03.local\_blast\_clustered\_putefflist\_to\_pres-abs\_table.py om te kijken wat de vorige ontwikkelaar precies gedaan heeft. Hieruit kwam de conclusie het script veel redudantie en recursie had; het was beter om een nieuwe aanpak te gebruiken om de aanwezigheid van effectoren op andere genomen te bepalen.

Eerst werden databases van elk genoom gemaakt. Dit werd gedaan met het command “makeblastdb”. Hier werd de volgende parameter aan meegegeven: “-dbtype nucl”. Deze parameter geeft aan dat de input file uit nucleotiden bestaat, dus het database type moet ook uit nucleotiden bestaan.

Nadat van elk genoom een database was gemaakt, werden de gevonden effectoren geBLAST tegen de genomen. Het BLASTen was met het volgende command: “blastn -db database/databasenaam -query effectoren.fasta -evalue 0.001 -out result.out”. Waarbij de parameter -db aangaf welke database gebruikt werd, vervolgens gaf -query aan tegen welke file er geBLAST was, in dit geval tegen de geclusterde effectoren. De E-value gaf aan hoeveel hits er verwacht kunnen worden van vergelijkbare kwaliteit die toevallig worden gevonden. Hoe kleiner de E-value, hoe beter de gevonden matches zijn. In dit geval is de E-value op 0.001 gezet. De laatste parameter is -out, en het geeft aan naar welk bestand de output weggeschreven was.

Daarna werd de blast output geparsed, zodat er een dictionary gemaakt werd met als key de signaalpeptidesequentie van de effector, en als value een lijst met in welke genomen de effector voorkwam. Dit werd gedaan door te kijken in de BLAST output en te kijken of een match een hogere identity had dan de identity threshold (default: 30), ook werd gekeken of de hit een hogere E-value had dan de E-value threshold, Als dit het geval was, werd er gekeken of die signaalpeptidesequentie al in de dictionary als key zat, als dit niet het geval was, werd deze toegevoegd met de genoom naam waarmee een hit was gevonden. Als de genoom naam er al wel in zat dan werd de genoom naam toegevoegd aan de value van die key.

Vervolgens werd de score bepaald van elke effector. Als een effector wel in een genoom zat dan werd dit aangeven met een 1. Als deze er niet in zat dan werd dit aangeven met een 0. Tot slot werd alles in een tab-seperated file gezet.

## Clustering script:

*Materiaal:*

* Python 3.6;
* Biopython 3.6
* PC met Linux omgeving;
* NCBI-blast+ versie 2.10.0;
* 02.cluster\_putefflists.py( (bijlage 3)

*Methoden:*

Voor het maken van dit script werd gekeken naar hoe het bestaande script eruitzag. Dit script was erg chaotisch, omdat niks in een functie stond. Ook was er veel informatie die meegenomen werd vanuit de blast die niet nodig was voor het verdere clusteren.

In het script werd eerst een BLAST-database gemaakt. Deze database werd gemaakt van alle effectoren die in de voorgaande stappen zijn gevonden. Vervolgens waren alle sequenties geblast tegen deze database. Elke sequentie zal altijd zichzelf vinden. Op het moment dat een sequentie ook genoeg lijkt op een andere sequentie werd deze ook in het cluster gedaan. Er was bij het bepalen van de matches, die door blast gevonden werden, gefilterd op de percentage\_identity en op length\_threshold parameters. Dit waren beiden parameters die in het config-bestand aangepast konden worden. Percentage\_identity was het percentage overeenkomst tussen de twee sequenties. De standaardwaarde hiervoor was 60% overeenkomst.

Wanneer de blastparen waren bepaald, werden door middel van single-linkage clusters gemaakt. Dit stuk van het script is grotendeels overgenomen van het al bestaande script omdat dit goed werkte. Er werden een paar aanpassingen gedaan om het werkend te krijgen in Python 3.6 en er is commentaar aan toegevoegd, zodat het duidelijk is wat er gebeurd.

In het oude script werd er vervolgens in het cluster gezocht naar de langste sequentie en deze werd vervolgens weggeschreven. Deze optie is nog steeds ingebouwd in het script. Er is ook een extra optie toegevoegd; in de config file kan aangegeven worden om een consensussequentie te maken van het cluster. Wanneer dit iss aangegeven gaat het script een ander deel van de code in en wordt de consensus sequentie bepaald van het cluster. Dit is bepaald doormiddel van een functie van Biopython.

Als laatste stap in het script wordt de langste sequentie of de consensussequentie van de gevonden clusters weggeschreven. Het wordt weggeschreven in een fasta file.

## 

## 04.cluster\_and\_plot\_heatmap3.R. script:

*Materiaal:*

* R versie 3.5;
* R package gplots;
* R package extrafont;
* R package ade4;
* R package bioLite ctc;
* PC met Linux omgeving;
* Heatmap3.R van github (<https://gist.github.com/nachocab/3853004>);
* 04.cluster\_and\_plot\_heatmap3.R (bijlage 5).

*Methoden:*

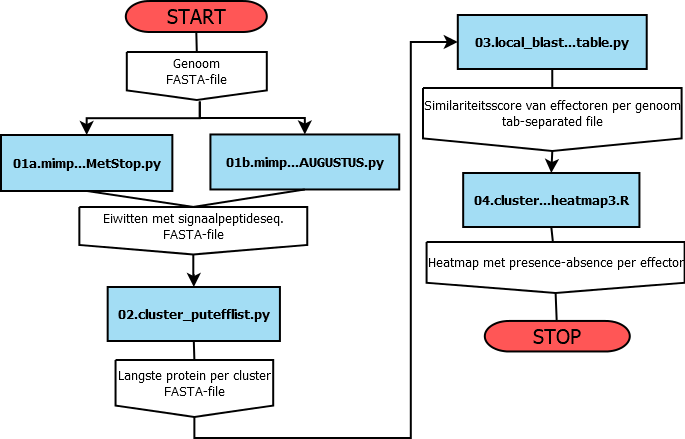
Eerst was gekeken hoe de vorige ontwikkelaar het script 04.cluster\_and\_plot\_heatmap3.R had gemaakt. De conclusie was dat het script 04.cluster\_and\_plot\_heatmap3.R een klein en goed script was, dus het script Heatmap3.R kon gebruikt worden.

Vervolgens werd gekeken of de output van onze data gelezen kon worden door de scripts 04.cluster\_and\_plot\_heatmap3.R en Heatmap3.R. De outputs van de scripts kwamen overeen met de verwachtingen. Het enige wat verkeerd was, was de plaats waar de output werd neergezet. Dit werd veranderd door het script in de Snakefile aan te roepen, en als argument de directory mee te nemen waar de output geplaatst moest worden.

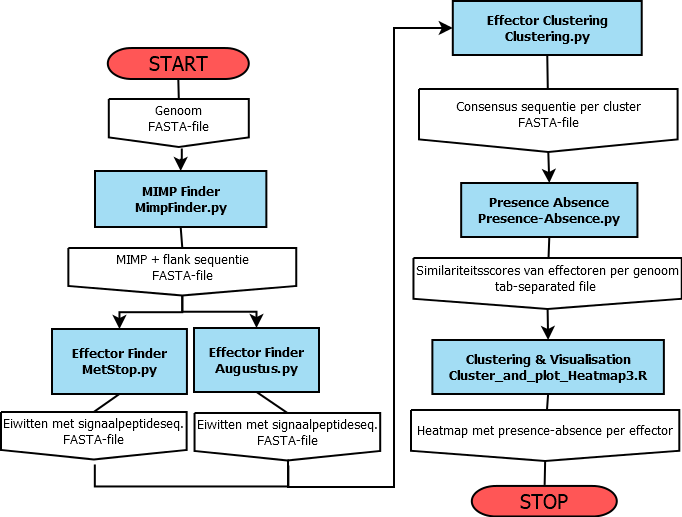
# Resultaten

## Flowchart oude en nieuwe pipeline

In Figuur 3 is de oude flowchart te zien en in Figuur 4 de nieuwe flowchart. De flowcharts beginnen met “START” en volgen een pad naar “STOP”. De witte blokken zijn de input en/of output data. De blauwe blokken zijn de modules en/of de scripts die de input gebruiken en output creëren.



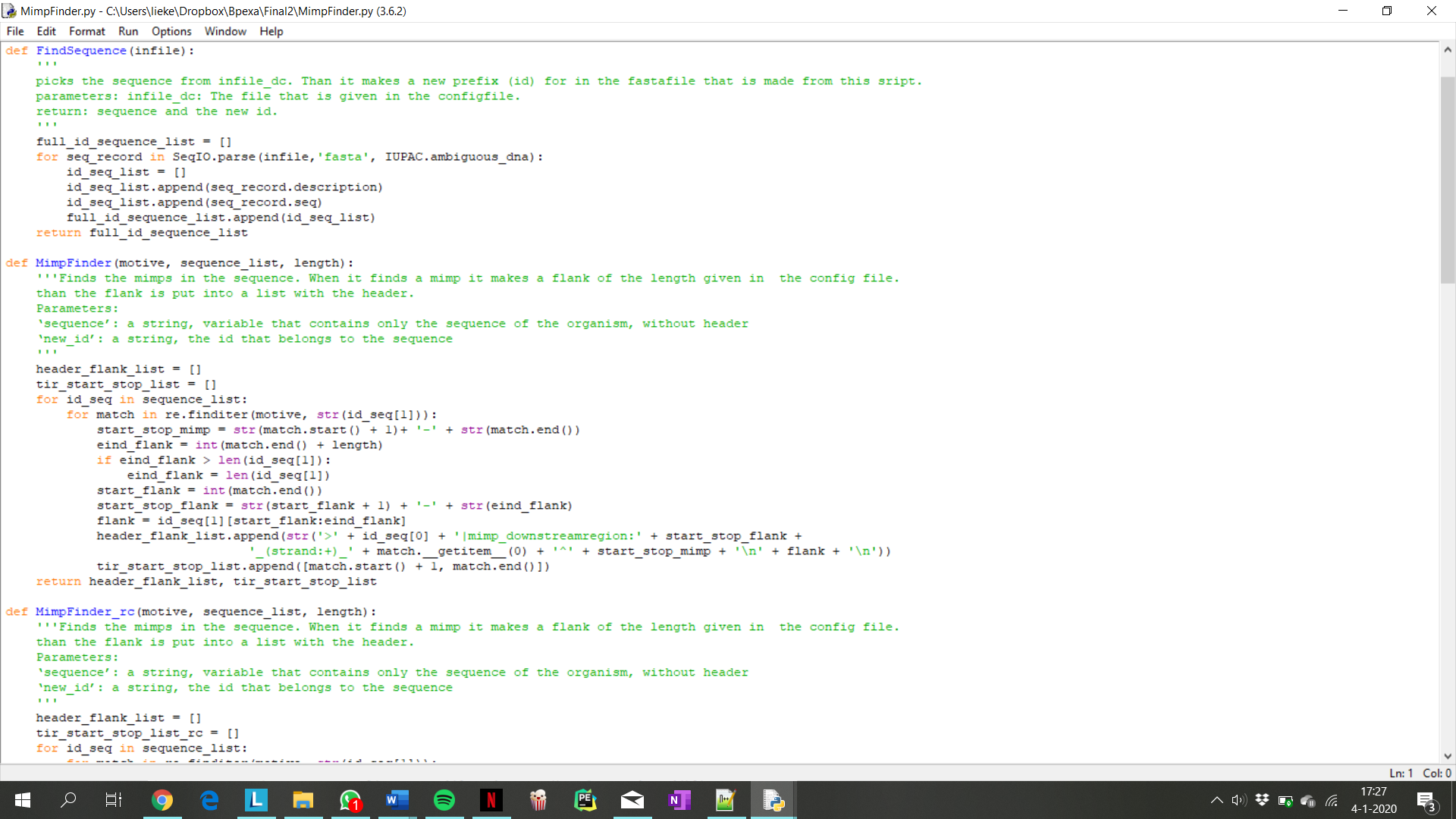
**Figuur 3.** *In de Figuur is een flowchart van de oude pipeline te zien. De flowchart begint bij start en eindigt bij stop. De wit gekleurde blokken zijn in/ouput files en de blauw gekleurde blokken zjin modules of script die worden gebruikt.*



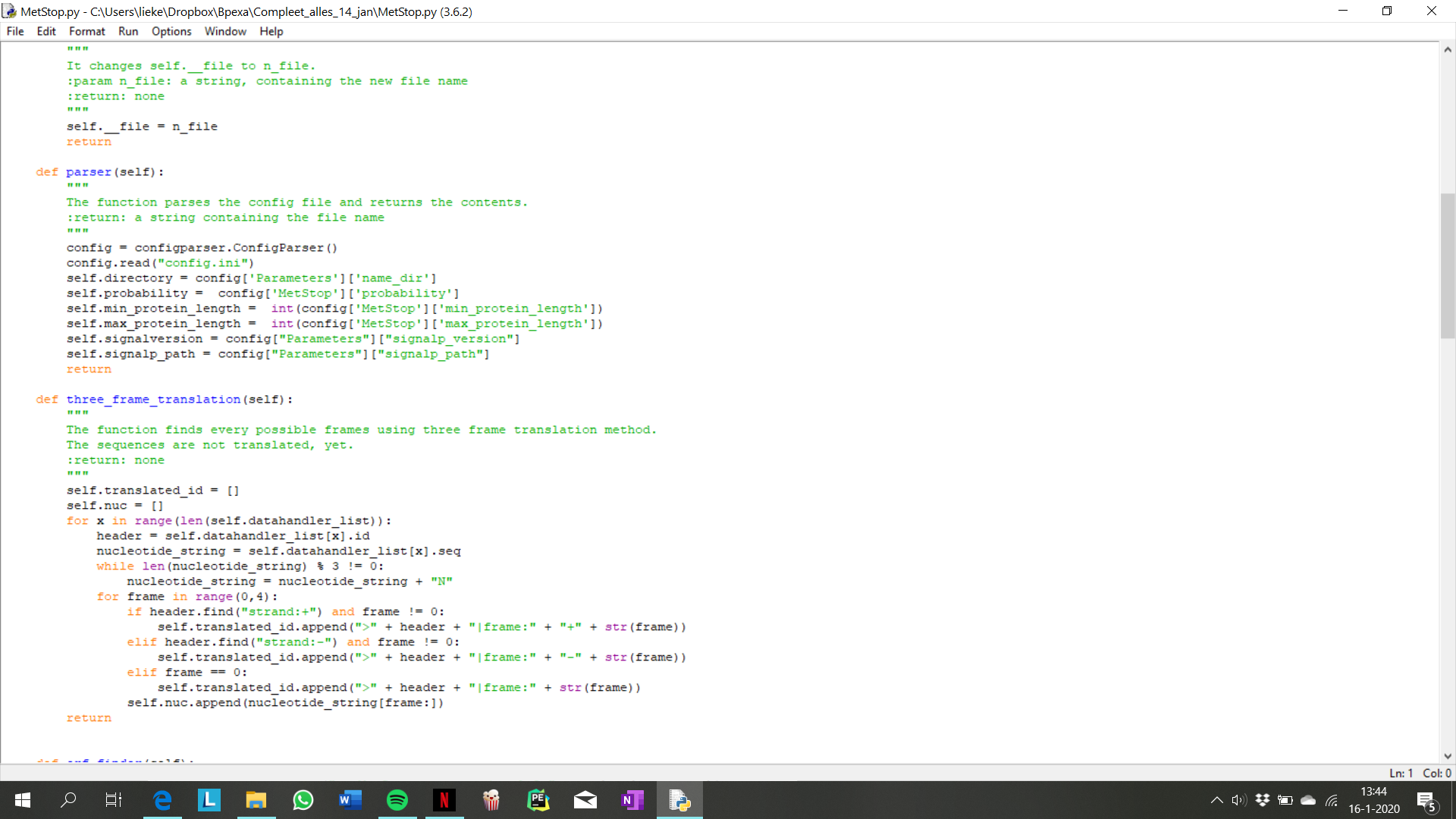
**Figuur 4.** *In de Figuur is een flowchart van de nieuwe pipeline te zien. De flowchart begint bij start en eindigt bij stop. De wit gekleurde blokken zijn in/ouput files en de blauw gekleurde blokken zjin modules of script die worden gebruikt.*

## Nieuwe pipeline en extra bestanden

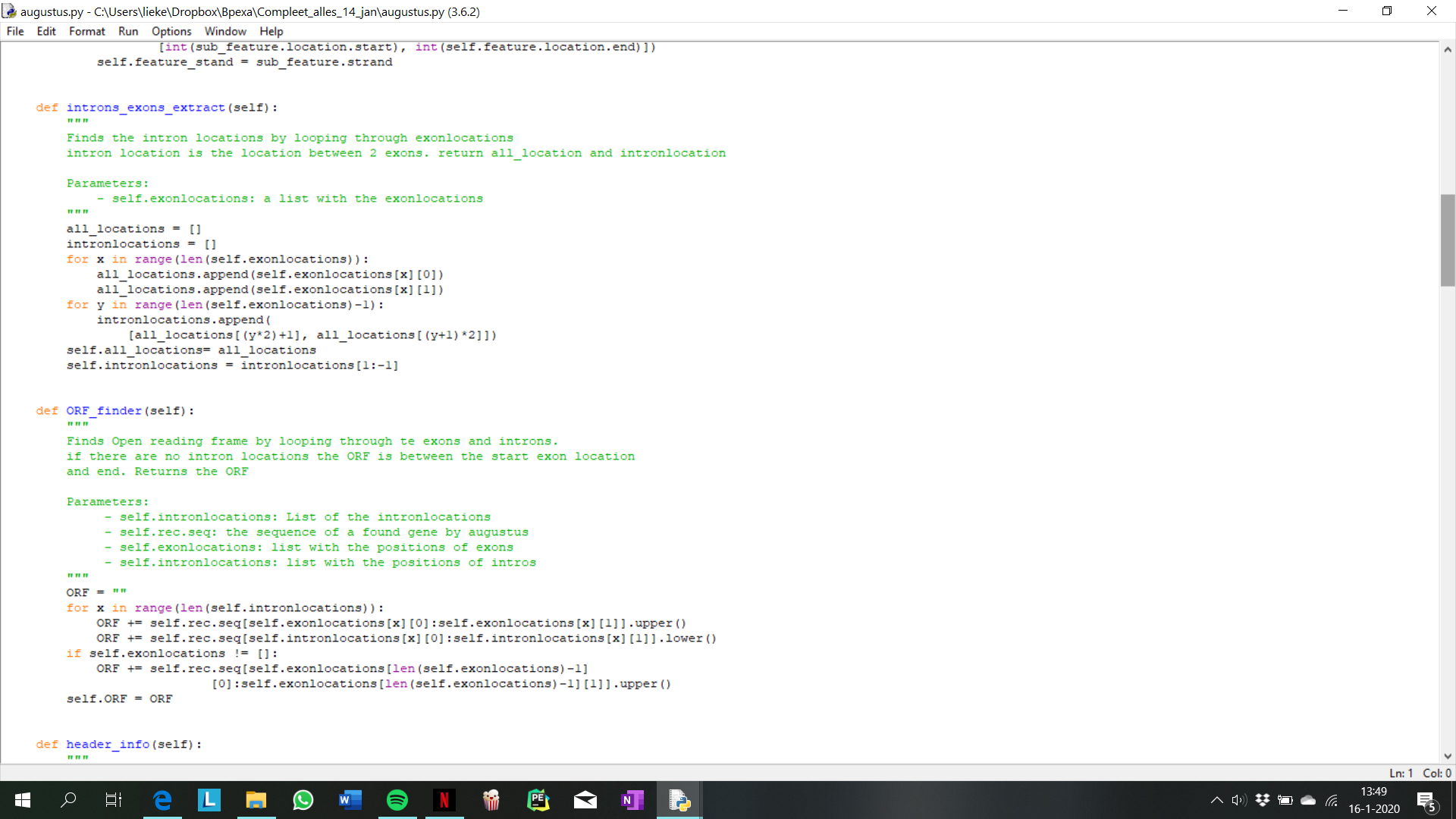
In deze sectie worden de nieuwe pipeline en de belangrijkste bestanden weergegeven en besproken.

Er is een nieuwe en verbeterde pipeline geschreven. In Figuur 5 tot 8 zijn delen van het script van respectievelijk MimpFinder.py, MetStop.py, augustus.py, presence\_absence.py en clustering.py te zien. In bijlage 7 tot en met 11 is de volledige code te zien van deze scripts. 

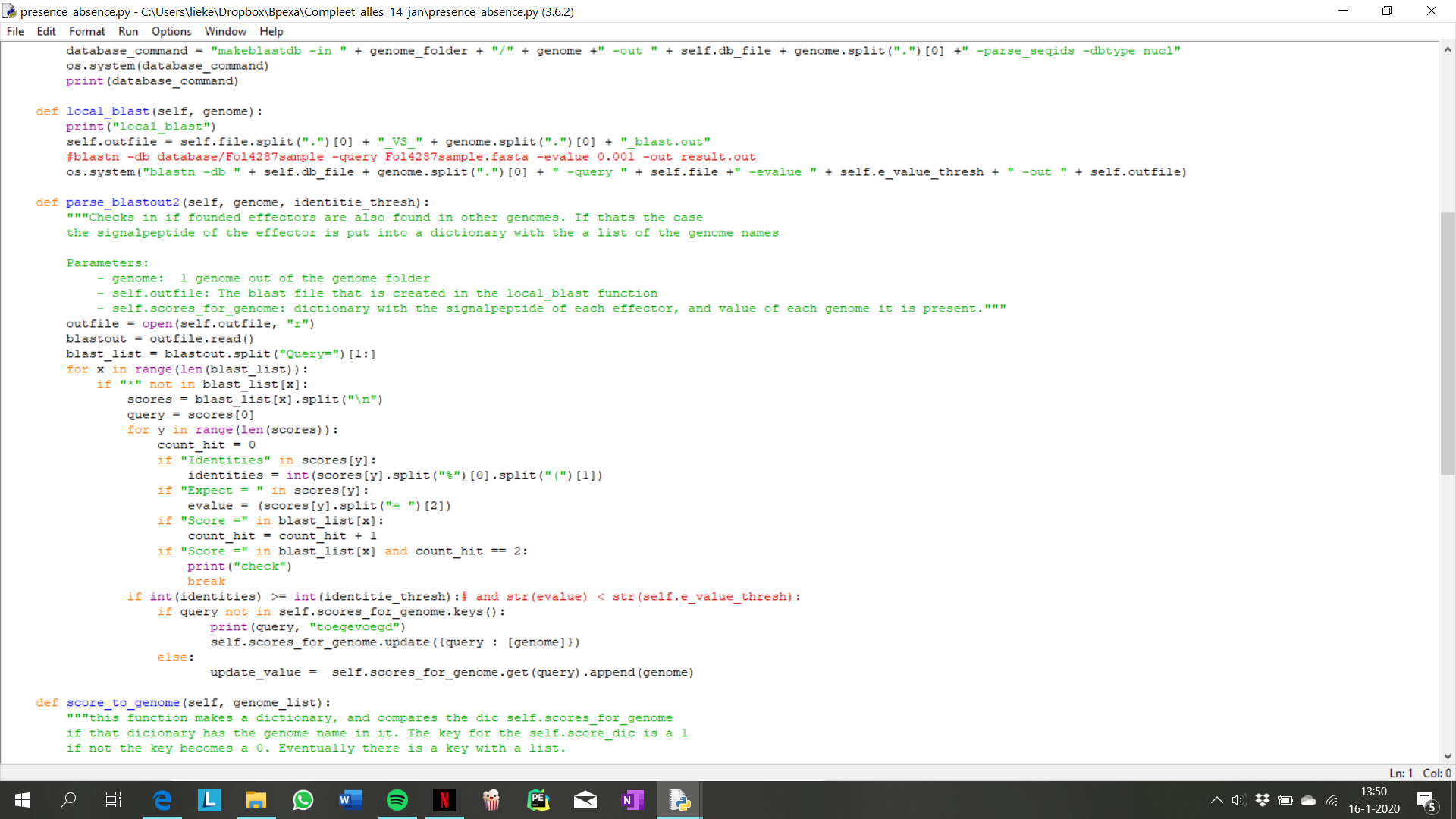
**Figuur 5.** *In de Figuur is een deel van het MimpFinder script weergegeven..*



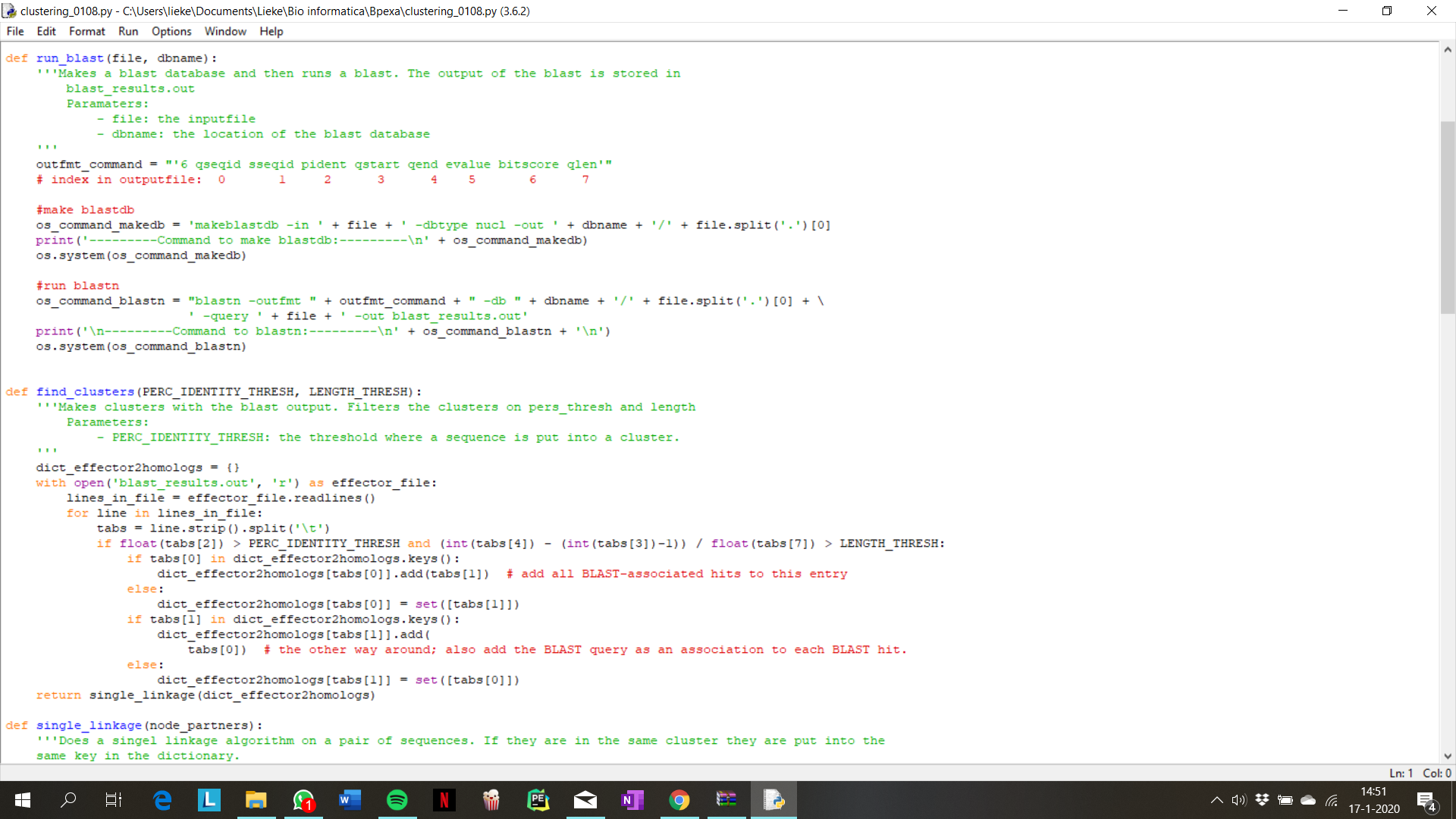
**Figuur 6.** *In de Figuur is een deel van het EffectorFinder script, MetStop.py, te zien.*



**Figuur 7.** *In de Figuur is een deel van het EffectorFinder script, augustus.py, te zien.*



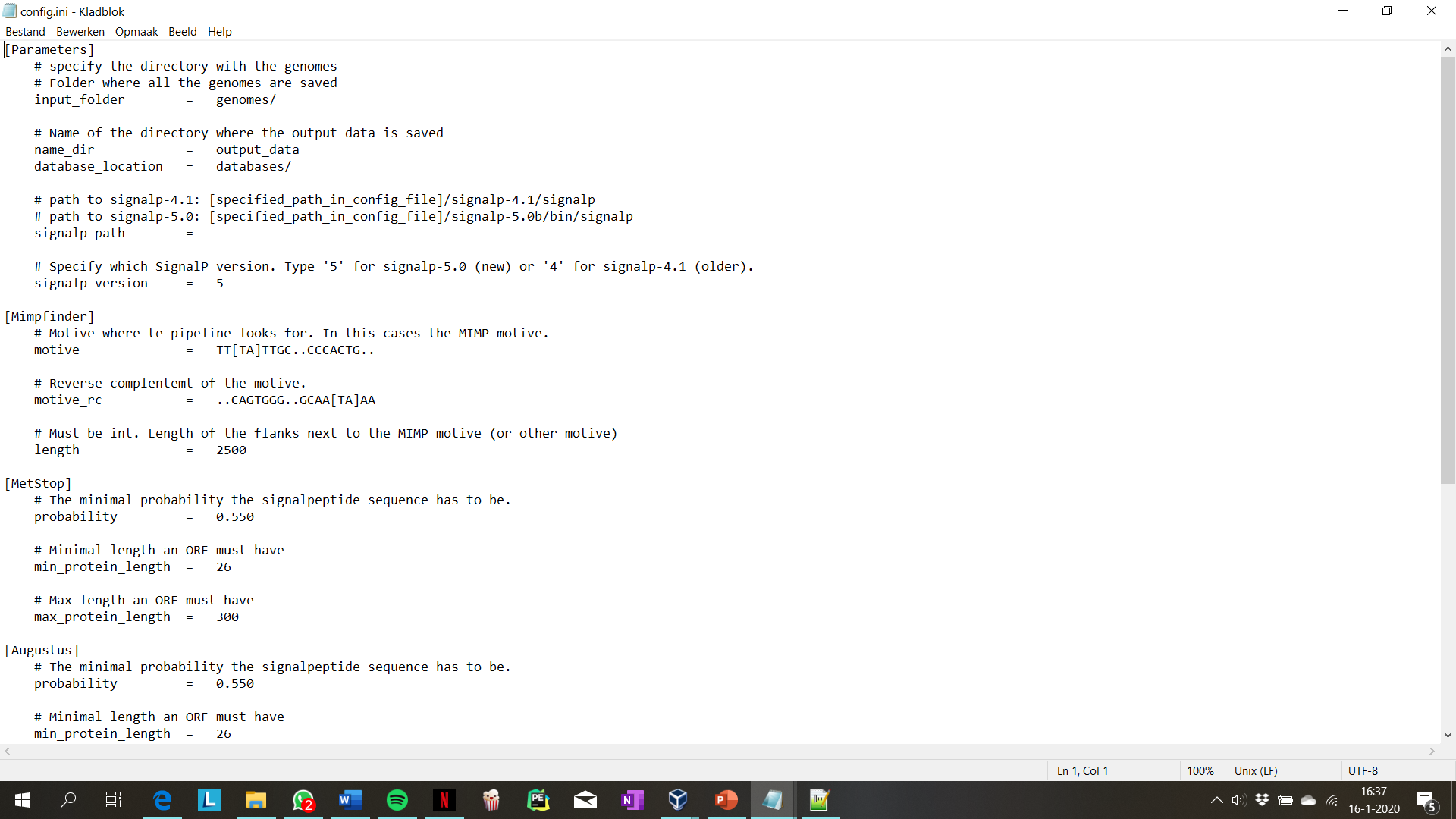
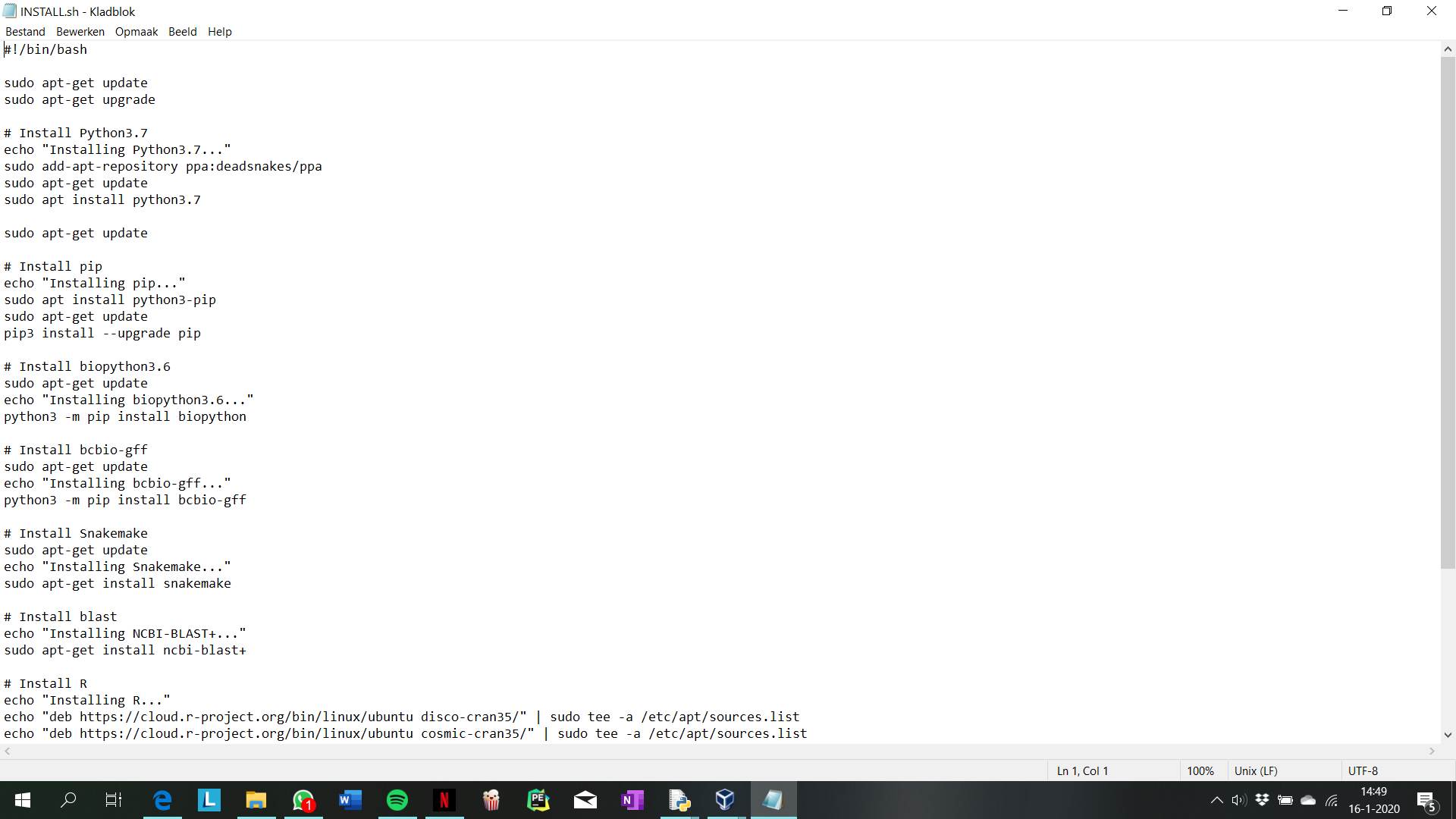
**Figuur 8.** *In de Figuur is een deel van het presence\_absence.py script te zien.*



**Figuur 9.** *In de Figuur is een deel van het clustering script, clustering.py, te zien.*

Voordat het script gerund kan worden zijn er een aantal programma’s die geïnstalleerd moeten worden. Hiervoor is een bash script geschreven die de benodigde programma’s installeert (voor zover dat mogelijk is), namelijk INSTALL.sh. Dit script kan later eventueel ook gebruikt worden om een Docker container te maken. Een aantal programma’s moeten handmatig geïnstalleerd of gedownload worden, zoals SignalP en R packages. Dit staat beschreven in de README file. In Figuur 10 is een deel van het install script te zien, in bijlage 12 is het volledige script weergegeven.

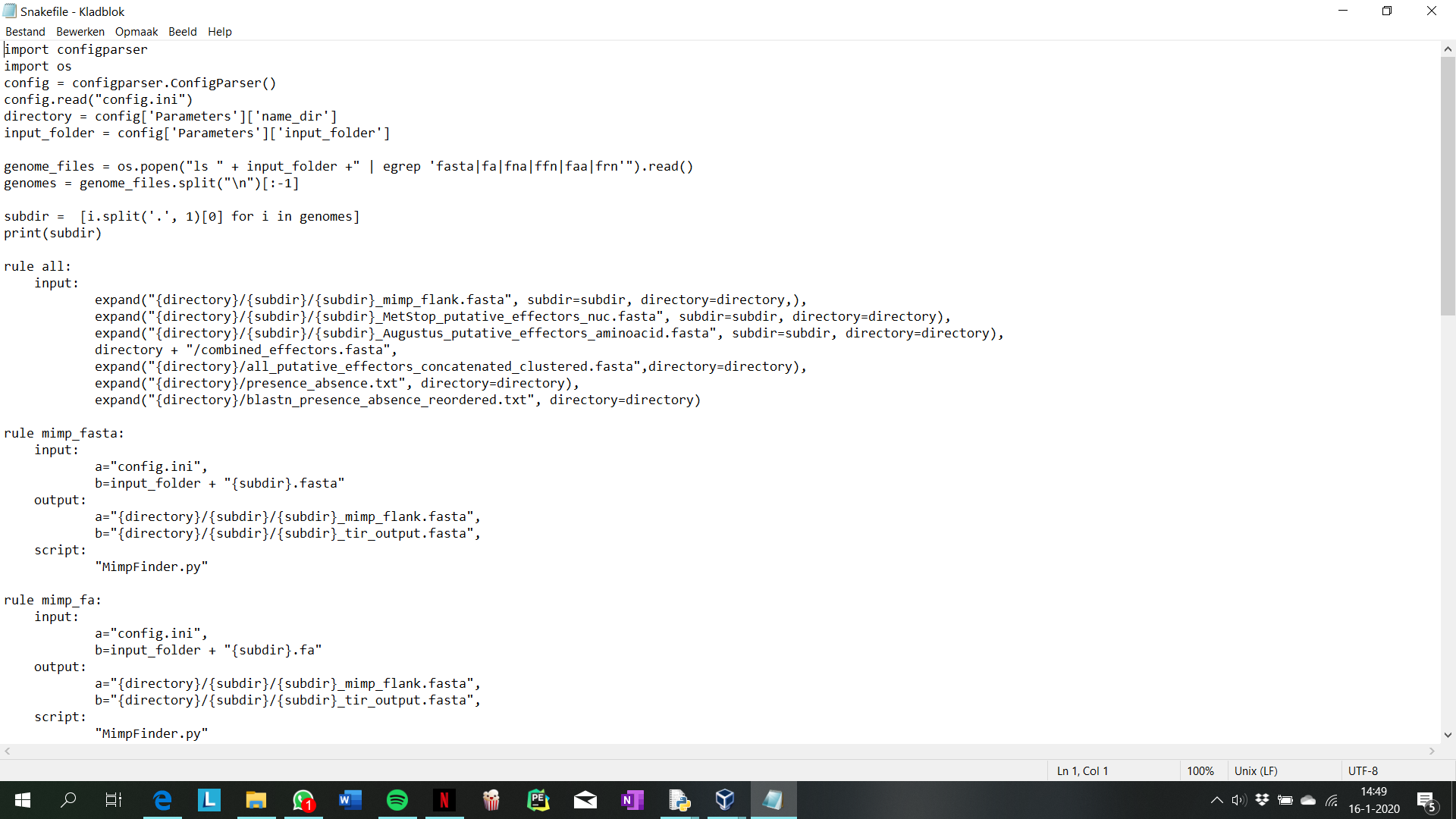
**Figuur 10.** *In de Figuur is het begin van het INSTALL.sh script te zien.*



**Figuur 11.** *In de Figuur is het begin van de config file te zien.*

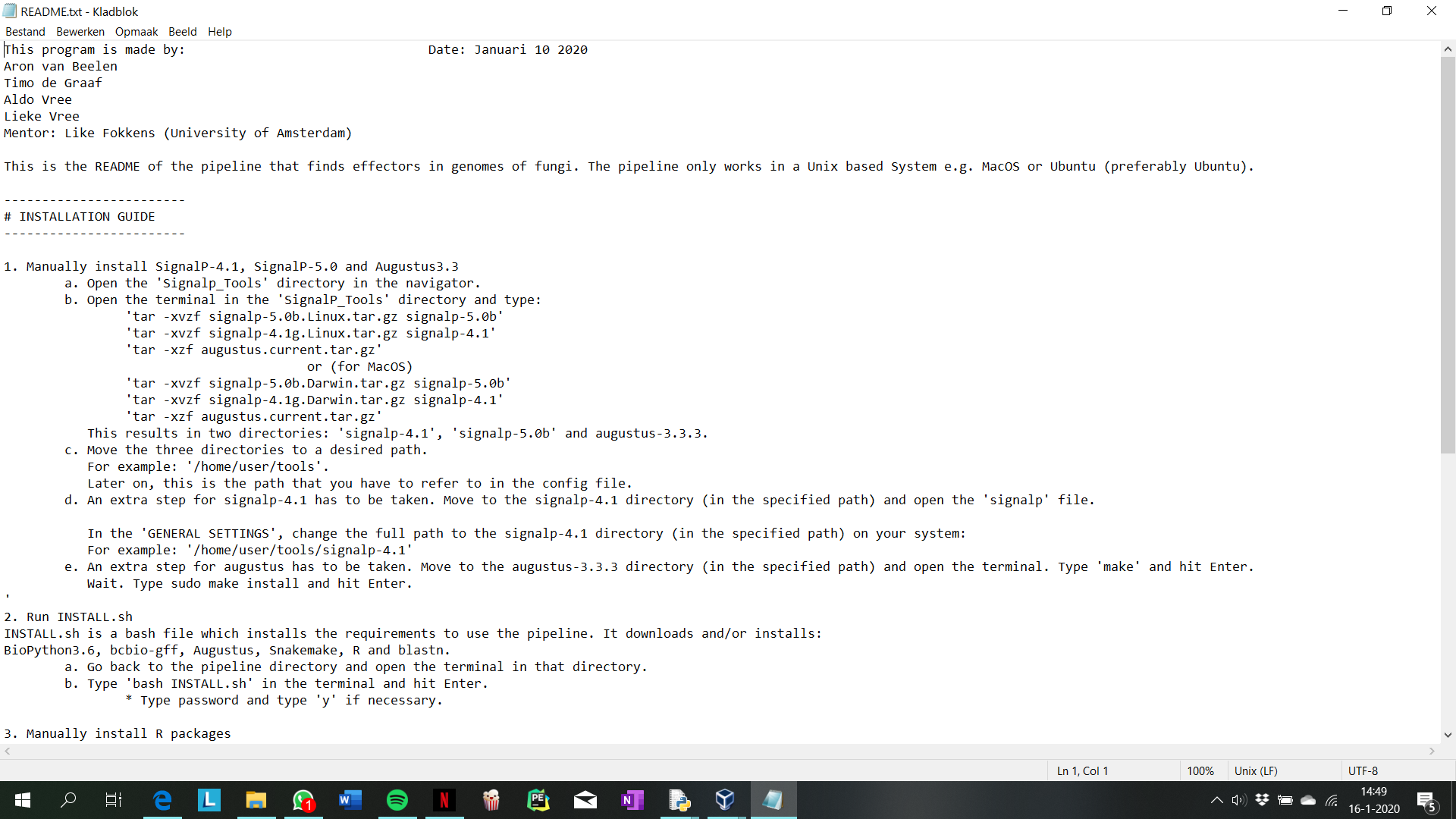
Ook is een config file geschreven, waarin alle verschillende parameters aangepast kunnen worden. Het begin van de config file is te zien in Figuur 11, de volledige config file is gegeven in bijlage 13.

Vervolgens is er een snakefile gemaakt. Deze is deels te zien in Figuur 12 en volledig in bijlage 14.



**Figuur 12.** *In de Figuur is het begin van de snakefile te zien.*

Tenslotte is een README file geschreven, README.txt. In deze README wordt eerst stap voor stap uitgelegd welke acties ondernomen moeten worden om de juiste programma’s te installeren in het onderdeel ‘INSTALLATION GUIDE’. Vervolgens staat beschreven hoe de pipeline gebruikt moet worden in het onderdeel ‘PIPELINE USAGE GUIDE’. In Figuur 13 is het begin van de README file te zien, in bijlage 15 is de volledige README te lezen.



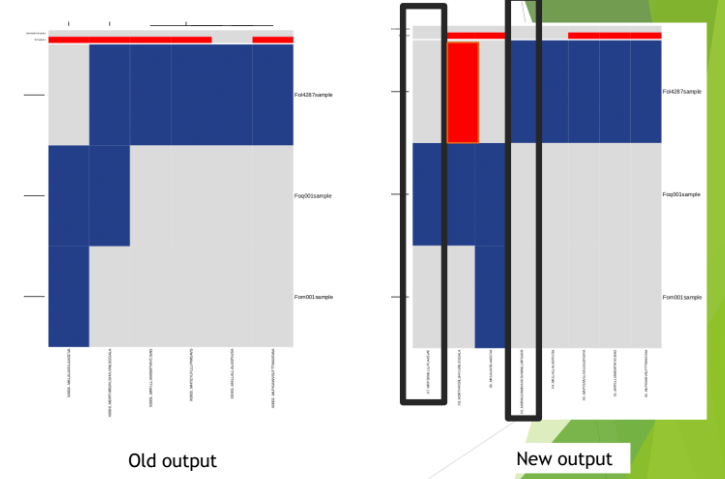
**Figuur 13.** *In de Figuur is het begin van de README file te zien.*

De laatste output van het script is de heatmap. In Figuur 14 is de output van het oude script naast de output van het nieuwe script te zien. Er is een verschil in output, in het nieuwe script worden twee nieuwe effectoren gevonden (de twee zwarte rechthoeken in Figuur 14). Ook is één effector bij de nieuwe output maar op een genoom aanwezig, waar deze bij de oude output nog op twee genomen aanwezig was (rode blok in Figuur 14), deze effector is “MDRTHTGRLSHVMLSGGALA”. Deze effector wordt maar op één genoom gevonden omdat bij het blasten van de effector tegen de genomen er alleen een hit gevonden wordt bij het genoom van Foq001 en niet bij Fol487.

*Figure SEQ Figure \\* ARABIC 13*

## 

**Figuur 14.** *Visualisatie van de geclusterde effectoren van de nieuwe en oude code. De effectoren met de zwarte omlijsting zijn niet gevonden door de oude output, en het rode blokje betekend dat 1 effector niet aanwezig is op een genoom, waarbij hij bij de oude output wel aanwezig is.*



## Belangrijkste verschillen

Er zijn een aantal dingen verbeterd aan de pipeline. Het belangrijkste was om de pipeline overzichtelijker en beter aanpasbaar te maken. De oude pipeline was erg onoverzichtelijk geschreven. De functies waren erg lang, er werden bijvoorbeeld vaak for loops in for loops in for loops geschreven. Er was niet echt één duidelijk doel per functie, maar een functie deed veel verschillende dingen. Ook was er veel redundantie. Een voorbeeld van het oude script was eerder te zien in Figuur 1. Dit maakt het erg lastig om de code aan te passen of nieuwe functies toe te voegen. Ook is te zien dat de regels van de code veel te lang zijn.

In de nieuwe code is ervoor gezorgd dat functies los van elkaar geschreven worden en de functies niet te veel of te lange regels bevatten, zodat de leesbaarheid beter is. Er is op gelet dat één functie ook maar één doel heeft. Op deze manier kan in een functie een andere functie aangeroepen worden en krijg je niet te veel loops in loops. Ook is het nu makkelijker om de code aan te passen, omdat je datgene wat je wilt aanpassen makkelijker kunt vinden in de code, en is het makkelijker om een functie toe te voegen. In Figuur 5 tot en met 8 zijn voorbeelden van de nieuwe code te zien.

Daarnaast bevatte de oude code weinig en incompleet commentaar. Ook stond dit commentaar vaak midden in het script en niet per functie. In de nieuwe code is gelet op goede documentatie van de functies. Per functie is een uitgebreide functiebeschrijving gegeven (in het Engels), worden de parameters die in het script worden gebruikt beschreven en wordt aangegeven wat de functie teruggeeft. Het commentaar is volgens het zogenaamde reStructuredText format geschreven, zodat dit beter geparsed kan worden door bijvoorbeeld sphinx.

Een ander belangrijk verschil is dat de versies van de gebruikte programma’s geupgrade zijn ten opzichte van het oude script. Het oude script is geschreven in Python versie 2.7 en maakte gebruik van SignalP versie 4.1 en Augustus versie 3.3.1. Het nieuwe script is geschreven in Python versie 3.6 en maakt gebruik van SignalP versie 5 en Augustus versie 3.3.3.

Ook is gebruik gemaakt van een snakefile zodat scripts parallel aan elkaar kunnen runnen. Dit zorgt ervoor dat het script sneller klaar zal zijn. In het oude script moesten de verschillende variabelen en paths in het script zelf aangepast worden. Om niet te hoeven zoeken in het script en deze variabelen makkelijk aan te kunnen passen is een config file gemaakt, waarin alle verschillende variabelen en paths opgegeven kunnen worden. Daarnaast is een install.sh file gemaakt, waarbij alle programma’s die nodig zijn om de pipeline te runnen automatisch worden geïnstalleerd. In de oude pipeline moesten alle programma’s handmatig worden geïnstalleerd.

In Tabel 1 is een samenvattend overzicht te zien van alle punten die zijn verbeterd ten opzichte van het oude script.

Tabel 1: *In de Tabel is een samenvattend overzicht te zien van alle verbeterpunten. De verbeterpunten zijn verdeeld in onderdelen, die in de eerste kolom staan. Vervolgens wordt in de tweede kolom beschreven hoe dit onderdeel was in het oude script. Tenslotte wordt in de laatste kolom beschreven hoe dit onderdeel in het nieuwe script is.*

|  |  |  |
| --- | --- | --- |
| **Onderdeel** | **Het oude script** | **Het nieuw script** |
| Functies | Grote functies, met te lange regels. Onoverzichtelijk en veel loops in loops. | Functies van ± 15 regels en niet te lange regels. Losse functies. |
| Documentatie | Weinig commentaar bij functies en random tussen code geplaatst. | Per functie een uitgebreide functiebeschrijving volgens reStructuredText format. |
| Code | Code procedureel geschreven | Code objectgeoriënteerd geschreven. |
| Tools | Python versie 2.7, Augustus 3.3.1 en SignalP versie 4.1. | Python versie 3.6, Augustus 3.3.3 en SignalP versie 4.1 en 5 (eigen keuze). |
| Installatie | Installatie instructies, handmatig alle benodigde programma’s installeren. | Bash file die alle benodigde programma’s installeert. (Alleen voor Ubuntu) |
| Configuratie | Handmatig parameters aanpassen in het script. | Config file waar alle voorkeuren en parameters aangegeven kunnen worden. |
| Extra | - | Snakefile voor parallellisatie. |

## Performance test

In Tabel 2 zijn de resultaten van de performance test weergegeven. De performance test is achtereenvolgens uitgevoerd op één systeem om bias te minimaliseren. De performance test is uitgevoerd met SignalP versie 4.1, omdat dit de versie is die gebruikt werd in de oude pipeline en er dan een betere vergelijking gemaakt kan worden. De gemiddelde runtime van de oude pipeline is 63,9 seconden met een standaarddeviatie van 1,1 en de nieuwe pipeline 41,9 seconden met een standaarddeviatie van 1,3. Het gemiddelde van het procentueel verschil is -34,4%.

Tabel 2: *In de Tabel is de perfomance test met data en statistieken te zien. In kolom 1 het aantal runs van 1 t/m 5, kolom 2 de runtijd in seconden van de oude pipeline, kolom 3 de runtijd in seconden van de nieuwe pipeline en kolom 4 het procentueel verschil van de runtijd van de oude en nieuwe pipeline. Daarnaast is het gemiddelde en de standaardeviatie onderaan de Tabel weergegeven.*

|  |  |  |  |
| --- | --- | --- | --- |
| **Aantal keren gerund** | **Runtime oude pipeline (seconden)** | **Runtime nieuwe pipeline (seconden)** | **Procentueel verschil (%)** |
| 1 | 65,5 | 41,4 | -36,8 |
| 2 | 62,5 | 42,0 | -32,9 |
| 3 | 63,9 | 39,9 | -37,6 |
| 4 | 64,6 | 42,9 | -33,6 |
| 5 | 63,2 | 43,6 | -31,0 |
| **Gemiddeld (X̄)** | 63,9 | 41,9 | -34,4 |
| **Standaarddeviatie (*S*)** | 1,1 | 1,3 |  |

# Discussie

## Parallellisatie, Modularisatie en Documentatie

De pipeline is er qua leesbaarheid en gebruiksvriendelijkheid erg op vooruit gegaan. Functies zijn niet te lang en het is niet meer één lange code, maar de code is opgesplitst in losse functies en de scripts zijn in klassen geschreven. Dit maakt het een stuk makkelijker om de code aan te passen en eventueel functies toe te voegen. Daarnaast zijn alle functies uitgebreid gedocumenteerd volgens het reStructuredText format. Ook is een config file gemaakt, zodat de parameters niet meer in de code zelf aangepast hoeven te worden, dit scheelt een hoop zoekwerk en tijd. Ook is er een installatiebestand gemaakt die alle benodigde programma’s installeert (voor zover mogelijk), zodat de gebruiker dit niet allemaal handmatig hoeft te doen. Er is een README bestand geschreven die een installatie handleiding en een pipeline handleiding bevat, hierin worden uitgebreid alle stappen uitgelegd om de benodigde programma’s te installeren en de pipeline te runnen. Ten slotte, is Snakemake gebruikt om een snakefile te creëren die parallellisatie mogelijk maakt. De pipeline kan achtereenvolgens meerdere input genomen verwerken.

## Performance test

De performance test is betrouwbaar, want de performance test is vijf keer gerund per pipeline op één systeem. De standaarddeviatie is voor beide pipelines laag (oud: 1,1; nieuw: 1,3) dit betekent dat de spreiding van de datapunten zeer klein is; de runs zijn betrouwbaar genoeg om deze datapunten mee te nemen in verdere statistische berekeningen, zoals het gemiddelde. De gemiddelde runtime voor de oude pipeline is 63,9 seconden en de nieuwe pipeline 41,9 seconden. Het procentueel verschil is -34,4%, dus de nieuwe pipeline is 34,4% sneller dan de oude pipeline. Dit is een fors verschil.

## Nieuwe versies van gebruikte tools

De versies van de gebruikte programma’s en tools zijn geüpdatet, namelijk Python 2.7 naar 3.6, SignalP 4.1 naar 5.0 en Augustus 3.3.1 naar 3.3.3. SignalP versie 5.0 werkt niet op alle systemen; om te voorkomen dat het script niet te gebruiken is op deze systemen is het script zo geschreven dat de gebruiker er ook nog steeds voor kan kiezen om gebruik te maken van SignalP versie 4.1. De code is daarnaast op zo’n manier geschreven het ook mogelijk is om nieuwe versie toe te voegen, mocht dat nodig zijn, omdat de code makkelijker aan te passen is.

## Input genoom

De oude en nieuwe pipeline maken gebruik van een FASTA-bestand met een FO-genoom. FO is een schimmel, dus een eukaryoot; eukaryotische genen bevatten intronen, dit betekent dat ORF’s gevonden worden met mogelijk intronen. Dus, niet alle ORF’s zijn betrouwbaar en kunnen intronen bevatten die, in werkelijkheid, niet gecodeerd worden.

## Oude pipeline met testdata

Tijdens het verbeteren van de oude pipeline is gebruik gemaakt van de testdata die de ontwikkelaar ook gebruikte. De reden hiervoor is om de output van de nieuwe pipeline te vergelijken en te valideren op de output van de oude pipeline. Gedurende het project werd de code te veel aangepast aan de testdata, waardoor de pipeline niet altijd goed functioneel was op andere input data. Een voorbeeld is dat de headers in de testdata een lengte hadden die kleiner was dan 100. Bij andere data met een header groter dan een lengte van 300 werkte het script niet. Dit komt voornamelijk door hard coding en het niet robuust maken van de scripts. Ook de integratie van scripts en hun input/output was niet goed gecoördineerd, waardoor soms input/output data verkeerd werd bewerkt.

## Heatmaps

Een verrassend verschil met de output van de oude en de nieuwe pipeline is dat de heatmaps net iets van elkaar verschillen. Deze verschillen kunnen verklaard worden. Ten eerste is een nieuwe versie van SignalP gebruikt. Deze nieuwe versie is beter in het voorspellen van signaalpeptides die vervolgens worden gekoppeld aan een mogelijke effector. De oude versie van SignalP vond geen signaalpeptide die gekoppeld kon worden aan die effector, daarom vindt de oude versie 1 effector minder. De andere extra effector wordt waarschijnlijk gevonden omdat het MetStop.py script herschreven is en het script daardoor iets gevoeliger en beter in het herkennen van effectoren is. Tot slot kan de effector die bij het nieuwe script maar op één genoom gevonden wordt in plaast van twee verklaard worden doordat het oude script gebruik maakt van de Python module “Bio Blast”. De nieuwe code blast met behulp van de command-line. Het kan zijn dat hier verschil in zit waardoor de oude code de effector wel op 2 genomen vindt.

## Toekomst

Verschillende scripts moeten volledig gedebugd worden, zodat het werkt op alle data. Hier moeten dan ook nieuwe testfuncties voor geschreven worden. Ook moet de README compleet gemaakt worden, want het voldoet niet aan de regels van een readme; een projectomschrijving en het doel van de pipeline mist, maar hier kan eventueel de oude README voor gebruikt worden.

Voor het maken van de heatmap is het R-script van de vorige ontwikkelaar gebruikt, het script is goed genoeg, maar het kan in de toekomst herschreven worden zodat het nog gebruiksvriendelijker wordt. De output zou veelzijdiger kunnen worden, zodat je er meer informatie uit kunt halen. Het script zou dan zo geschreven kunnen worden dat je de keuze hebt tot meer visualisatie mogelijkheden. Zoals bijvoorbeeld een echte heatmap met verschillende kleurintensiteiten.

Ook zou een docker container gemaakt kunnen worden, zodat een omgeving gemaakt wordt waarin de code gerund kan worden, die hetzelfde is als waarin de code is getest. Dit om te voorkomen dat er verschillen komen in output door verschillen in implementatie. Ook kan bijvoorbeeld SignalP 5 niet op alle systemen goed geïnstalleerd worden, dit is dan ook geen probleem meer. Er moet echter wel gekeken worden of dit mogelijk is met de rechten van SignalP.

Ten slotte kunnen er nieuwe functies aan het script worden toegevoegd, zoals een functie die niet alleen MIMPs vindt maar alle repeats in een genoom.

# Conclusie

De oude pipeline is verbeterd op het gebied van modularisatie, documentatie en parallellisatie: De pipeline is modulairder gemaakt door middel van korte, leesbare functies die één doel ondersteunen en de scripts zijn herschreven in klassen om een gemodulariseerde structuur te creëren. De pipeline heeft een verbeterde documentatie, want de functies en klassen zijn duidelijk gedocumenteerd. Ook is het README bestand verbeterd die simpeler uitlegt hoe de gebruiker de pipeline gebruikt en de benodigde software installeert en hiervoor is ook een install file gemaakt, om het nog makkelijker te maken voor de gebruiker. De pipeline is geparallelliseerd met behulp van Snakemake, zodat meerdere genomen tegelijk bewerkt kunnen worden. De nieuwe pipeline is daarnaast ook een stuk sneller dan de oude pipeline. De pipeline is echter nog niet helemaal bug vrij en werkt niet altijd even goed op alle data.

# Referentielijst

# Köster J, Rahmann S. Snakemake-a scalable bioinformatics workflow engine. *Bioinformatics.* 2012; 28(19): 2520-2522. doi: https://doi.org/10.1093/bioinformatics/bts480

# van Dam P, Fokkens L, Schmidt SM, et al. Effector profiles distinguish formae speciales of Fusarium oxysporum. *Environmental Microbiology*. 2016; 18(11): 4087-4102. doi: 10.1111/1462-2920.13445.

# Horizontal Gene Transfer. Horizontal Gene Transfer - an overview | ScienceDirect Topics. https://www.sciencedirect.com/topics/neuroscience/horizontal-gene-transfer. Accessed January 18, 2020.

# Michielse CB, Rep M. Pathogen profile update: Fusarium oxysporum. *Mol Plant Pathol.* 2009; 10(3): 311-324. doi: 10.1111/j.1364-3703.2009.00538.x.

# Schimdt SM, Houterman PM, Schreiver I, Ma L, Amyotte S, Chellappan B et al. MITEs in the promoters of effector genes allow prediction of novel virulence genes in Fusarium oxysporum. *BMC Genomics*. 2013; 14: 119. doi: 10.1186/1471-2164-14-119.

# Bijlagen

## Bijlage 1 - 01a.mimpfinder\_combine\_to\_putefflist\_MetStop.py

from datetime import datetime

startTime = datetime.now()

import sys, os, re, glob

from Bio import SeqIO

from Bio.Seq import Seq

from Bio.SeqRecord import SeqRecord

from Bio.Alphabet import generic\_dna

from Bio.Alphabet import IUPAC

def MimpFinder(infile, sc\_prefix, motiefje, motiefje\_rc, datahandler, distance, mimpsequencesfile, infilename):

datahandler\_list = []

nrofcompletemimps = 0

complete\_mimp\_counter = 0

nrofincompletemimps = 0

open(mimpsequencesfile,'w').close()

print '-'\*20

for seq\_record in SeqIO.parse(infile, 'fasta', IUPAC.ambiguous\_dna):

print '// Analyzing \'%s\'... (%i bp long)' % (seq\_record.description, len(seq\_record))

if str(sc\_prefix) in seq\_record.description:

new\_id = seq\_record.description.split(sc\_prefix)[1].split('\_')[0] #this splitting of '\_' is necessary for SignalP in order to not skip all queries in datahandler3 (because their headers might contain '\_').

if ' ' in new\_id: #if 'contig\_xx' is not the last phrase in the fasta header, resulting in ne\_id to become 'contig\_xx whole shotgun sequencing bladiebla'

new\_id = new\_id.split(' ')[0]

else:

print '-'\*20 + '\nThis supercontig prefix was not found; add supercontig prefix as 1st argument..!\n' + '-'\*20

quit()

match = re.finditer(motiefje, str(seq\_record.seq))

match\_rc = re.finditer(motiefje\_rc, str(seq\_record.seq))

ir\_dict = {}

region\_to\_find\_TIR\_mate =''

i=0

n=0

for m in match:

if m.end()+distance >= len(seq\_record): #ensure that no value bigger than the supercontig length will arise

end\_pos\_after\_mimp = len(seq\_record)

else:

end\_pos\_after\_mimp = m.end()+distance

#ir\_dict 0=start position, 1=end position, 2=motif sequence^location, 3=ORF-search-sequence

ir\_dict[i] = [m.start()+1, m.end(), seq\_record.seq[m.start():m.end()], seq\_record.seq[m.end():end\_pos\_after\_mimp]]

region\_record = SeqRecord(seq=ir\_dict[i][3],

id='sc' +new\_id+ '| '+ 'mimp\_downstreamregion:' + str(ir\_dict[i][1]+1) + '\_' + str(end\_pos\_after\_mimp) + ' (strand: +)'+ ' ' + str(ir\_dict[i][2])+'^'+str(m.start()+1)+'-'+str(m.end()),

description='')

datahandler\_list.append(region\_record)

print ' >' + region\_record.id

i+=1

if m.start()-400 > 0:

region\_to\_find\_TIR\_mate\_coord1 = m.start()-400

region\_to\_find\_TIR\_mate\_coord2 = m.start()

region\_to\_find\_TIR\_mate = seq\_record.seq[region\_to\_find\_TIR\_mate\_coord1:region\_to\_find\_TIR\_mate\_coord2]

else:

region\_to\_find\_TIR\_mate\_coord1 = 0

region\_to\_find\_TIR\_mate\_coord2 = m.start()

region\_to\_find\_TIR\_mate = seq\_record.seq[region\_to\_find\_TIR\_mate\_coord1:region\_to\_find\_TIR\_mate\_coord2]

match\_TIR\_mate = re.finditer(motiefje\_rc, str(region\_to\_find\_TIR\_mate))

mimp\_completeness = False

mimp\_fasta\_record = ''

for TIR\_mate in match\_TIR\_mate:

mimp\_completeness = True

location\_of\_mimp\_coord1=region\_to\_find\_TIR\_mate\_coord1+TIR\_mate.start()

location\_of\_mimp\_coord2=m.end()

mimp\_sequence=seq\_record.seq[location\_of\_mimp\_coord1:location\_of\_mimp\_coord2]

if mimp\_completeness:

nrofcompletemimps+=0.5 # 0.5 will also be added in the reverse complement part of the script

complete\_mimp\_counter+=1

mimp\_fasta\_record = '>'+infilename+'\_mimp\_'+str(complete\_mimp\_counter)+' \_contig\_'+new\_id+'('+str(location\_of\_mimp\_coord1)+':'+str(location\_of\_mimp\_coord2)+')\n'+str(mimp\_sequence)+'\n'

with open(mimpsequencesfile,'a') as mimpsequencesfilewriter:

mimpsequencesfilewriter.write(mimp\_fasta\_record)

else:

nrofincompletemimps+=1

for m in match\_rc:

if m.start() <= distance: #ensure that no negative value will arise

start\_pos\_before\_mimp = 0

else:

start\_pos\_before\_mimp = m.start()-distance

ir\_dict[n] = [m.start()+1, m.end(), seq\_record.seq[m.start():m.end()], seq\_record.seq[start\_pos\_before\_mimp:m.start()]]

region\_record = SeqRecord(seq=ir\_dict[n][3].reverse\_complement(), #make RC of selected area of 2000bp

id='sc' +new\_id+ '| '+ 'mimp\_upstreamregion:' + str(start\_pos\_before\_mimp) + '\_' + str(ir\_dict[n][0]-1) + ' (strand: -, rev. transc.)'+ ' ' + str(ir\_dict[n][2].reverse\_complement())+'^'+str(m.start()+1)+'-'+str(m.end()),

description='')

datahandler\_list.append(region\_record)

print ' >' + region\_record.id

n+=1

if m.end()+400 < len(seq\_record.seq):

region\_to\_find\_TIR\_mate = seq\_record.seq[m.end():m.end()+400]

else:

region\_to\_find\_TIR\_mate = seq\_record.seq[m.end():len(seq\_record.seq)]

match\_TIR\_mate = re.finditer(motiefje, str(region\_to\_find\_TIR\_mate))

mimp\_completeness = False

for TIR\_mate in match\_TIR\_mate:

mimp\_completeness = True

if mimp\_completeness:

nrofcompletemimps+=0.5

else:

nrofincompletemimps+=1

print 'nrofincompletemimps (1 TIR):', nrofincompletemimps, 'nrofcompletemimps (2 TIRs):', int(nrofcompletemimps)

print '// Motif found: ' + str(len(datahandler\_list)) + 'x.'

SeqIO.write(datahandler\_list, datahandler, 'fasta')

nrofcompletemimps=int(nrofcompletemimps)

print '\n// Wrote %s %sbp regions downstream of mimp IR motif to %s' % (len(datahandler\_list), distance, datahandler)

return nrofcompletemimps, nrofincompletemimps

def Translator(infile, datahandler2, distance):

datahandler\_list2 = []

for seq\_record in SeqIO.parse(infile, 'fasta', IUPAC.ambiguous\_dna):

for i in range(3): #(i=0 - i=1 - i=2)

s = seq\_record.seq[i:]

while len(s)%3 != 0: # add N to the end of the region if not devisable by 3

s += 'N'

translated\_record = SeqRecord(seq=s.translate(table=1), id=seq\_record.id, description=seq\_record.description+' frame='+str(i))

datahandler\_list2.append(translated\_record)

SeqIO.write(datahandler\_list2, datahandler2, 'fasta')

print '-'\*20

print '// Wrote %s translated %iaa entries to %s' % (len(datahandler\_list2), (distance/3), datahandler2)

print '-'\*20

def OrfFinder(datahandler2, min\_prot\_len, datahandler3, orfs, max\_prot\_len, max\_d2m):

for seq\_record in SeqIO.parse(datahandler2, 'fasta', IUPAC.protein):

genomic\_region\_up = int(seq\_record.description.split('region:')[1].split('\_')[0])

genomic\_region\_down = int(seq\_record.description.split('region:')[1].split('\_')[1].split(' ')[0])

strand = seq\_record.description.split('strand:')[1][1]

frame = int(seq\_record.description.split('frame=')[1][0])

mimpseq = seq\_record.description.split(') ')[1].split(' ')[0]

#print '\n'+genomic\_region\_up + ' - ' +genomic\_region\_down + ' ' +strand

MetStop(seq\_record.seq, seq\_record.id, genomic\_region\_up, genomic\_region\_down, strand, frame, mimpseq, min\_prot\_len, datahandler3, orfs, max\_prot\_len, max\_d2m)

SeqIO.write(orfs, datahandler3, 'fasta')

print '// Wrote %s protein sequences with a mimp IR motif in their promoter and an ORF >%i and <%i aa to %s' % (len(orfs), min\_prot\_len, max\_prot\_len, datahandler3)

print '-'\*20

def MetStop(sequence, ident, genomic\_region\_up, genomic\_region\_down, strand, frame, mimpseq123, min\_prot\_len, datahandler3, orfs, max\_prot\_len, max\_d2m):

met\_location = sequence.find('M')

stop\_location = met\_location+1

while met\_location >= 0 and stop\_location>=0:

stop\_location = sequence.find('\*', met\_location+1)

prot = sequence[met\_location:stop\_location+1] #+1 = add STOP

dist\_to\_mimp = (met\_location\*3)+frame #genomic distance

dist\_to\_mimp\_fromstop = (stop\_location\*3)+frame+2

if len(prot) > min\_prot\_len:

#print 'M:' + str(met\_location) + '; \*:' + str(stop\_location) + ': '+ prot

if strand == '+':

startpos = genomic\_region\_up+dist\_to\_mimp

endpos = genomic\_region\_up+dist\_to\_mimp\_fromstop

orf\_record = SeqRecord(seq=prot.strip('\*'), id=str(ident) +str(startpos)+'-'+str(endpos)+'|d2m:'+str(dist\_to\_mimp)+'bp|'+strand+'|f'+str(frame)+'|len:'+str(len(prot)-1), description=mimpseq123)

if len(prot) < int(max\_prot\_len):

if int(dist\_to\_mimp) < max\_d2m:

orfs.append(orf\_record)

elif strand == '-':

endpos = genomic\_region\_down-dist\_to\_mimp

startpos = genomic\_region\_down-dist\_to\_mimp\_fromstop

orf\_record = SeqRecord(seq=prot.strip('\*'), id=str(ident) +str(startpos)+'-'+str(endpos)+'|d2m:'+str(dist\_to\_mimp)+'bp|'+strand+'|f'+str(frame)+'|len:'+str(len(prot)-1), description=mimpseq123)

if len(prot) < int(max\_prot\_len):

if int(dist\_to\_mimp) < max\_d2m:

orfs.append(orf\_record)

met\_location = sequence.find('M', met\_location+1)

#return met\_location

def OrfWriter(datahandler3, signalpfile, min\_prot\_len, proteinoutfile, SignalPpath, SignalP\_threshold):

RunSignalP(datahandler3, signalpfile, 'euk', SignalPpath, SignalP\_threshold)

SPorfs = []

#Parse SignalP output file:

with open(signalpfile, 'r+') as sp:

for line in sp:

if line.startswith('Name='):

sp\_name = line.split('Name=')[1].split('\tSP=')[0]

orfs = SeqIO.parse(datahandler3, 'fasta', IUPAC.protein)

for orf in orfs:

if orf.id == sp\_name:

sp\_present = line.split('\tSP=\'')[1].split('\'')[0] # SP: YES OR NO

orf.id += '|SP=' + sp\_present

orf.id += ';D='+line.split(' D=0')[1].split(' ')[0] # SP D-value

if sp\_present == 'YES':

cleavage\_site\_pos1 = line.split('Cleavage site between pos. ')[1].split(' and ')[0]

cleavage\_site\_pos2 = line.split('Cleavage site between pos. ')[1].split(' and ')[1].split(': ')[0]

if 'X' not in (orf.seq[:int(cleavage\_site\_pos2)-1]): #prevent 'NNN' translated to 'X' in signalpeptides to end up in the list

signalpeptideseq = str(orf.seq[:int(cleavage\_site\_pos2)-1])

orfie = SeqRecord(seq=signalpeptideseq.lower()+orf.seq[int(cleavage\_site\_pos2)-1:].upper(), id=orf.id, description=orf.description.split(' ')[1]+'\_signalpeptideseq='+signalpeptideseq)

SPorfs.append(orfie)

sp.close()

print ' Total # of sequences matching the criteria: %i' % len(SPorfs)

#Write all protein sequences that meet requirements (close to motif, longer than 30 aa and contains signal peptide) to proteinoutfile:

SeqIO.write(SPorfs, proteinoutfile, 'fasta')

def RunSignalP(datahandler3, signalpfile, organism, SignalPpath, SignalP\_threshold):

print '// Running SignalP 4.1...'

cline = SignalPpath+' -t %s -f summary -u %s %s > %s' % (organism, SignalP\_threshold, datahandler3, signalpfile)

os.system(cline)

def ExtractOrfToFasta(proteinsfasta, uberinfile, puteff\_dnaseqs, genome, puteff\_logfile, combined\_puteff\_fasta, combined\_puteff\_logfile, filecounter, nrofcompletemimps, nrofincompletemimps, combined\_puteff\_logfile2, sc\_prefix):

#EXTRACT GENOMIC SEQUENSE:

open(puteff\_dnaseqs, 'wb').close() #clear genomic DNA sequence fastafile

dnaoutfile = open(puteff\_dnaseqs, 'a')

open(puteff\_logfile, 'wb').close()

logheader="genome\tputeff\_supercontig\tgenomic\_start\_pos\tgenomic\_end\_pos\tdist2mimp\torientation\tprotlength\tD\_value\tmimp\_IR\_seq\tmimp\_IR\_pos\tsignalpeptideseq\tproteinseq\tgenomicsequence\n"

logheader2="genome\tnr\_of\_complete\_mimps\tnr\_of\_incomplete\_mimps\tnr\_of\_put\_eff\_MetStop\n"

puteff\_logfile\_writer = open(puteff\_logfile, 'a')

puteff\_logfile\_writer.write(logheader) #for each genome, write a log with a header.

combined\_putefffile = open(combined\_puteff\_fasta, 'a')

combined\_puteff\_logfile\_writer = open(combined\_puteff\_logfile, 'a')

combined\_puteff\_logfile2\_writer = open(combined\_puteff\_logfile2, 'a')

if filecounter < 1: # During running of the first genome file, the log header should be added at the top in the case of a combined log file.

combined\_puteff\_logfile\_writer.write(logheader)

combined\_puteff\_logfile2\_writer.write(logheader2)

proteinsfastafile = SeqIO.parse(proteinsfasta, 'fasta')

n=0

print '// Putative effectors found in genome: ' + genome

for seq\_record in proteinsfastafile:

n+=1

puteff\_supercontig = seq\_record.description.split('sc')[1].split('|')[0]

#MAKE SURE THE CONTIG IN THE ORIGINAL FASTA FILE IS LIKE THIS; contig\_1 (space no comma behind it)

genomic\_start\_pos = int(seq\_record.description.split('|')[1].split('-')[0])

genomic\_end\_pos = int(seq\_record.description.split('|')[1].split('-')[1].split('|')[0])

dist2mimp = seq\_record.description.split('d2m:')[1].split('bp')[0]

orientation = seq\_record.description.split('bp|')[1].split('|')[0]

protlength = seq\_record.description.split('|len:')[1].split('|')[0]

D\_value = seq\_record.description.split('D=.')[1].split(' ')[0]

mimp\_IR\_seq = seq\_record.description.split(' ')[1].split('^')[0]

mimp\_IR\_pos = seq\_record.description.split('^')[1].split('\_signalpeptideseq=')[0]

signalpeptideseq = seq\_record.description.split('\_signalpeptideseq=')[1]

proteinseq = seq\_record.seq

all\_contigs = list(SeqIO.parse(uberinfile, "fasta", IUPAC.unambiguous\_dna))

for sc in all\_contigs:

sc\_id = sc.description.split(sc\_prefix)[1]

if "\_" in sc\_id:

sc\_id = sc\_id.split("\_")[0]

if sc\_id == puteff\_supercontig:

genomicsequence = sc.seq[genomic\_start\_pos-1:genomic\_end\_pos]

print ' contig\_'+str(puteff\_supercontig)+'\tposition '+str(genomic\_start\_pos)+'-'+str(genomic\_end\_pos)+'\t'+signalpeptideseq

putEff\_fastaentry = ">"+str(n).zfill(4)+'.'+signalpeptideseq+"\_"+genome+"\_d2m"+str(dist2mimp)+"\_len"+str(protlength)+"\n"+str(genomicsequence)+"\n\n"

dnaoutfile.write(putEff\_fastaentry)

puteff\_attributes = [genome, puteff\_supercontig, genomic\_start\_pos, genomic\_end\_pos, dist2mimp, orientation, protlength, D\_value, mimp\_IR\_seq, mimp\_IR\_pos, signalpeptideseq, proteinseq, genomicsequence]

putEff\_logentry = ('\t'.join(map(str,puteff\_attributes)))+'\n'

puteff\_logfile\_writer.write(putEff\_logentry)

#combined\_putefffile will collect all the output from the mimpsearch; this means there will be many redundant put effectors.

combined\_putefffile.write(putEff\_fastaentry)

combined\_puteff\_logfile\_writer.write(putEff\_logentry)

putEff\_logentry2 = genome+'\t'+str(nrofcompletemimps)+'\t'+str(nrofincompletemimps)+'\t'+str(n)+'\n'

combined\_puteff\_logfile2\_writer.write(putEff\_logentry2)

dnaoutfile.close() #collects inside genome out folder all DNA sequences of the putative effectors

puteff\_logfile\_writer.close() #writes a log for all puteff found in the current genome (inside genome out folder)

combined\_putefffile.close() #collects inside the out folder all DNA sequences of the putative effectors of all genomes that are being processed by the script.

combined\_puteff\_logfile\_writer.close() #writes a general log file with more details of the putative effectors identified

combined\_puteff\_logfile2\_writer.close()

print '-'\*20

print "// Finished with genome of %s; wrote %i genomic DNA sequences of putEff ORFs to %s" % (genome, n, puteff\_dnaseqs)

def MainDef(genomefastafile, directory, folder, combined\_puteff\_fasta, combined\_puteff\_logfile, filecounter, combined\_puteff\_logfile2, combined\_puteff\_dir):

#(default contig prefix)

sc\_prefix = sys.argv[3] #'contig\_'

infilename, infileextension = os.path.splitext(genomefastafile)

infile = directory+folder+'/'+genomefastafile

outdirectory = combined\_puteff\_dir+infilename+'\_mimpfinder\_out/'

if not os.path.exists(outdirectory):

os.makedirs(outdirectory)

mimpsequencesfile = outdirectory+infilename+'\_0\_mimpfinder\_completemimpsequences.fasta'

datahandler = outdirectory+infilename+'\_1\_mimpfinder\_downstreamregions.fasta'

datahandler2 = outdirectory+infilename+'\_2\_mimpfinder\_translateddownstreamregions.fasta'

datahandler3 = outdirectory+infilename+'\_3\_mimpfinder\_putativeORFs.fasta'

signalpfile = outdirectory+infilename+'\_4\_mimpfinder\_SignalP.summary\_out'

proteinoutfile = outdirectory+infilename+'\_5\_mimpfinder\_proteinseq\_out.fasta'

blastoutfile = outdirectory+infilename+'\_6\_mimpfinder\_blast\_out.csv'

puteff\_dnaseqs = outdirectory+infilename+'\_7\_mimpfinder\_puteff\_genomicseq\_out.fasta'

puteff\_logfile = outdirectory+infilename+'\_8\_mimpfinder\_puteff\_logfile.txt'

motiefje = 'TT[TA]TTGC..CCCACTG..'

motiefje\_rc = '..CAGTGGG..GCAA[TA]AA'

#motiefje = 'TT[TAC]TTGC[ACG][CTA]C[CT][CT]ACTG..' ## (Mara Bergemann, 2008)

#motiefje\_rc = '..CAGT[GA][GA]G[GAT][TGC]GCAA[TAG]AA'

orfs = []

distance = int(sys.argv[4]) # sequence downstream of motif used for ORF prediction

min\_prot\_len = int(sys.argv[5]) #in aa

max\_prot\_len = int(sys.argv[6])

max\_d2m = int(sys.argv[7]) # max distance between mimp TIR and start-codon

SignalPpath = str(sys.argv[8])

SignalP\_threshold = str(sys.argv[9])

nrofcompletemimps, nrofincompletemimps = MimpFinder(infile, sc\_prefix, motiefje, motiefje\_rc, datahandler, distance, mimpsequencesfile, infilename) #ir\_dict[i] = [m.start()+1, m.end(), seq\_record.seq[m.start():m.end()], seq\_record.seq[m.end():m.end()+distance]]

Translator(datahandler, datahandler2, distance)

OrfFinder(datahandler2, min\_prot\_len, datahandler3, orfs, max\_prot\_len, max\_d2m)

OrfWriter(datahandler3, signalpfile, min\_prot\_len, proteinoutfile, SignalPpath, SignalP\_threshold)

ExtractOrfToFasta(proteinoutfile, infile, puteff\_dnaseqs, infilename, puteff\_logfile, combined\_puteff\_fasta, combined\_puteff\_logfile, filecounter, nrofcompletemimps, nrofincompletemimps, combined\_puteff\_logfile2, sc\_prefix)

return mimpsequencesfile

if \_\_name\_\_ == "\_\_main\_\_":

######################

directory\_folder = sys.argv[1]

folder = sys.argv[1].split('/')[-1]

directory = directory\_folder.split(folder)[0]

output\_dir = sys.argv[2]

file\_extensions = (".fa", ".fasta", ".fas", "fna") # Specify the suffix of the genome files (.fa, .fasta, etc)

combined\_puteff\_dir = output\_dir+'/01.mimpfinder/'+folder+'\_MetStopOut/'

if not os.path.exists(combined\_puteff\_dir):

os.makedirs(combined\_puteff\_dir)

combined\_puteff\_fasta = combined\_puteff\_dir+'all\_putative\_effectors.fasta'

open(combined\_puteff\_fasta, 'w').close()

combined\_puteff\_logfile = combined\_puteff\_dir+'all\_putative\_effectors\_log.txt'

open(combined\_puteff\_logfile, 'w').close()

combined\_puteff\_logfile2 = combined\_puteff\_dir+'all\_putative\_effectors\_log2.txt'

open(combined\_puteff\_logfile2, 'w').close()

######################

mimpsequencesfile\_list = []

filecounter=0

for genomefastafile in os.listdir(directory\_folder):

if genomefastafile.endswith(file\_extensions):

print "\n// executing mimpfinder\_MetStop script for "+genomefastafile

mimpsequencesfile = MainDef(genomefastafile, directory, folder, combined\_puteff\_fasta, combined\_puteff\_logfile, filecounter, combined\_puteff\_logfile2, combined\_puteff\_dir)

mimpsequencesfile\_list.append(mimpsequencesfile)

filecounter+=1

else:

print '-'\*20

print '// THE END'

print "No more files with extension '%s' were found in directory '%s'" % (file\_extensions, (directory+folder))

print "Executed the script for %i files." % (filecounter)

print 'Total run time = %s' % (datetime.now()-startTime)

print '-'\*20

ConcatenateMimpSequences\_cmnd = 'cat '+' '.join(mimpsequencesfile\_list)+' > '+combined\_puteff\_dir+'all\_mimp\_sequences.fasta'

print ConcatenateMimpSequences\_cmnd, os.system(ConcatenateMimpSequences\_cmnd)

## Bijlage 2 - 01b.mimpfinder\_combine\_to\_putefflist\_AUGUSTUS.py

from datetime import datetime

startTime = datetime.now()

import sys, os, re

from Bio import SeqIO

from Bio.Seq import Seq

from Bio.SeqRecord import SeqRecord

from Bio.Alphabet import generic\_dna

from Bio.Alphabet import IUPAC

from Bio.SeqFeature import SeqFeature, FeatureLocation

from BCBio import GFF

def MimpFinder(infile, sc\_prefix, motiefje, motiefje\_rc, datahandler, distance):

datahandler\_list = []

print '-'\*20

for seq\_record in SeqIO.parse(infile, 'fasta', IUPAC.ambiguous\_dna):

print '// Analyzing \'%s\'... (%i bp long)' % (seq\_record.description, len(seq\_record))

if str(sc\_prefix) in seq\_record.description:

new\_id = seq\_record.description.split(sc\_prefix)[1].split('\_')[0] #this splitting of '\_' is necessary for SignalP in order to not skip all queries in datahandler3 (because their headers might contain '\_').

if ' ' in new\_id: #if 'contig\_xx' is not the last phrase in the fasta header, resulting in ne\_id to become 'contig\_xx whole shotgun sequencing bladiebla'

new\_id = new\_id.split(' ')[0]

else:

print '-'\*20 + '\nThis supercontig prefix was not found; add supercontig prefix as 1st argument..!\n' + '-'\*20

quit()

match = re.finditer(motiefje, str(seq\_record.seq))

match\_rc = re.finditer(motiefje\_rc, str(seq\_record.seq))

ir\_dict = {}

i=0

n=0

for m in match:

if m.end()+distance >= len(seq\_record): #ensure that no value bigger than the supercontig length will arise

end\_pos\_after\_mimp = len(seq\_record)

else:

end\_pos\_after\_mimp = m.end()+distance

#ir\_dict 0=start position, 1=end position, 2=motif sequence^location, 3=ORF-search-sequence

ir\_dict[i] = [m.start()+1, m.end(), seq\_record.seq[m.start():m.end()], seq\_record.seq[m.end():end\_pos\_after\_mimp]]

region\_record = SeqRecord(seq=ir\_dict[i][3],

id='contig' +new\_id+ '\_'+ 'mimp\_downstreamregion' + str(ir\_dict[i][1]+1) + '-' + str(end\_pos\_after\_mimp) + '\_strand1'+ '\_' + str(ir\_dict[i][2])+'^'+str(m.start()+1)+'-'+str(m.end()),

description='')

datahandler\_list.append(region\_record)

print ' >' + region\_record.id

###[comment these lines to not search \_upstream\_ of a forward motif:]###

# if m.start() <= distance: #ensure that no negative value will arise

# start\_pos\_before\_mimp = 0

# else:

# start\_pos\_before\_mimp = m.start()-distance

# ir\_dict[n] = [m.start()+1, m.end(), seq\_record.seq[m.start():m.end()], seq\_record.seq[start\_pos\_before\_mimp:m.start()]]

# region\_record = SeqRecord(seq=ir\_dict[n][3].reverse\_complement(), #make RC of selected area of 2000bp

# id='contig' +new\_id+ '\_'+ 'mimp\_upstreamregion' + str(start\_pos\_before\_mimp) + '-' + str(ir\_dict[n][0]-1) + '\_strand-1'+ '\_' + str(ir\_dict[n][2].reverse\_complement())+'^'+str(m.start()+1)+'-'+str(m.end()),

# description='')

# datahandler\_list.append(region\_record)

# print ' >' + region\_record.id

#######################################################################

i+=1

for m in match\_rc:

if m.start() <= distance: #ensure that no negative value will arise

start\_pos\_before\_mimp = 0

else:

start\_pos\_before\_mimp = m.start()-distance

ir\_dict[n] = [m.start()+1, m.end(), seq\_record.seq[m.start():m.end()], seq\_record.seq[start\_pos\_before\_mimp:m.start()]]

region\_record = SeqRecord(seq=ir\_dict[n][3].reverse\_complement(), #make RC of selected area of 2000bp

id='contig' +new\_id+ '\_'+ 'mimp\_upstreamregion' + str(start\_pos\_before\_mimp) + '-' + str(ir\_dict[n][0]-1) + '\_strand-1'+ '\_' + str(ir\_dict[n][2].reverse\_complement())+'^'+str(m.start()+1)+'-'+str(m.end()),

description='')

datahandler\_list.append(region\_record)

print ' >' + region\_record.id

###[comment these lines to not search \_upstream\_ of a forward motif:]###

# if m.end()+distance >= len(seq\_record): #ensure that no value bigger than the supercontig length will arise

# end\_pos\_after\_mimp = len(seq\_record)

# else:

# end\_pos\_after\_mimp = m.end()+distance

# #ir\_dict 0=start position, 1=end position, 2=motif sequence^location, 3=ORF-search-sequence

# ir\_dict[i] = [m.start()+1, m.end(), seq\_record.seq[m.start():m.end()], seq\_record.seq[m.end():end\_pos\_after\_mimp]]

# region\_record = SeqRecord(seq=ir\_dict[i][3],

# id='contig' +new\_id+ '\_'+ 'mimp\_downstreamregion' + str(ir\_dict[i][1]+1) + '-' + str(end\_pos\_after\_mimp) + '\_strand1'+ '\_' + str(ir\_dict[i][2])+'^'+str(m.start()+1)+'-'+str(m.end()),

# description='')

# datahandler\_list.append(region\_record)

# print ' >' + region\_record.id

#######################################################################

n+=1

print '// Motif found: ' + str(len(datahandler\_list)) + 'x.'

SeqIO.write(datahandler\_list, datahandler, 'fasta')

print '\n// Wrote %s %sbp regions downstream of mimp IR motif to %s' % (len(datahandler\_list),distance, datahandler)

#return ir\_dict

def PredictGenes(datahandler, datahandler2, datahandler3a, datahandler3b, datahandler3c, genome, max\_d2m, AUGUSTUS\_command, min\_prot\_len, max\_prot\_len):

ORF\_list = []

CDS\_list = []

protein\_list = []

genecounter = 0

print '// Running Augustus 3.1...'

print AUGUSTUS\_command

cline = AUGUSTUS\_command+' --species=fusarium --singlestrand=true %s > %s' % (datahandler, datahandler2) #--singlestrand=true

print(cline),os.system(cline)

in\_seq\_handle = open(datahandler)

seq\_dict = SeqIO.to\_dict(SeqIO.parse(in\_seq\_handle, "fasta"))

in\_seq\_handle.close()

in\_handle = open(datahandler2)

for rec in GFF.parse(in\_handle, base\_dict=seq\_dict):

print '>'+rec.id

feature\_ids =[]

for feature in rec.features:

if feature.id != '': feature\_ids.append(feature.id) # find out how many genes were predicted in this 2.5kb region

for feature in rec.features:

#print feature.strand

for feature\_id in feature\_ids:

if feature.id == feature\_id: # each gene is iterated in the gff file

print " "+"\_"\*20

print ' gene found:', feature.id, feature.location, feature.strand

#print feature.extract(rec).seq

CDS = '' ##

nrofexons = 0

exonlocations = []

for sub\_feature in feature.sub\_features:

#print sub\_feature.type, sub\_feature.location

if sub\_feature.type == 'CDS':

gene\_orientation = sub\_feature.strand # gene orientation compared to 2.5kb region > always with IR as 5' end

nrofexons +=1

if gene\_orientation == 1:

CDS+=sub\_feature.extract(rec).seq #add all exons together to form full CDS

else:

CDS+=sub\_feature.extract(rec).seq.reverse\_complement()

exonlocations.append([int(sub\_feature.location.start), int(sub\_feature.location.end)])

#if str(CDS[:3]) == 'ATG':

#print 'gene orientation:',gene\_orientation

#print 'exonlocations:', exonlocations

nrofintrons = nrofexons-1

all\_locations = []

if nrofexons > 1: #are there any introns?

for exonlocation in exonlocations:

all\_locations.append(exonlocation[0])

all\_locations.append(exonlocation[1]) #add coordinates of exons to one list all\_locations

intronlocations = []

for i in range(nrofintrons):

intronlocations.append([all\_locations[(i\*2)+1], all\_locations[(i+1)\*2]]) #extract the coordinates of each intron

ORF = '' ##

for e in range(len(intronlocations)):

ORF += rec.seq[exonlocations[e][0]:exonlocations[e][1]].upper()

ORF += rec.seq[intronlocations[e][0]:intronlocations[e][1]].lower()

ORF += rec.seq[exonlocations[len(exonlocations)-1][0]:exonlocations[len(exonlocations)-1][1]].upper()

contig = rec.id.split('contig')[1].split('\_mimp')[0]

region\_start = rec.id.split('region')[1].split('\_')[0].split('-')[0]

#print 'region\_start:', region\_start

region\_end = rec.id.split('region')[1].split('\_')[0].split('-')[1]

#print 'region\_end:', region\_end

strand = int(rec.id.split('\_strand')[1].split('\_')[0])

#print 'strand', strand

mimp\_IR\_seq = rec.id.split('\_strand')[1].split('\_')[1].split('^')[0]

mimp\_IR\_loc = rec.id.split('\_strand')[1].split('\_')[1].split('^')[1]

gene\_start\_relative = feature.location.start

gene\_end\_relative = feature.location.end

if gene\_orientation == 1:

if strand == 1:

real\_orientation = 1 #gene is found on coding strand downstream of MIMP IR

gene\_start\_absolute = int(region\_start)+int(gene\_start\_relative)

gene\_end\_absolute = int(region\_start)+int(gene\_end\_relative)-1

elif strand == -1:

real\_orientation = -1 #gene is found on other strand upstream of MIMP IR

gene\_start\_absolute = int(region\_end)-int(gene\_start\_relative)

gene\_end\_absolute = int(region\_end)-int(gene\_end\_relative)+1

elif gene\_orientation == -1:

ORF = ORF.reverse\_complement()

CDS = CDS.reverse\_complement()

if strand == 1:

real\_orientation = -1

gene\_start\_absolute = int(region\_start)+int(gene\_start\_relative)

gene\_end\_absolute = int(region\_start)+int(gene\_end\_relative)-1

elif strand == -1:

real\_orientation = 1

gene\_start\_absolute = int(region\_end)-int(gene\_start\_relative)

gene\_end\_absolute = int(region\_end)-int(gene\_end\_relative)+1

proteinseq = CDS.translate() ##

proteinseq = proteinseq[:-1] #trim off the STOP codon ('\*')

#new\_fastaheader = feature.id.split('.t')[0]+'\_'+genome+'\_contig'+contig+'\_'+str(gene\_start\_absolute)+'-'+str(gene\_end\_absolute)+'\_'+str(real\_orientation)+'\_d2m'+str(gene\_start\_relative)+'\_'+str(mimp\_IR\_seq)+'\_'+mimp\_IR\_loc+'\_'+str(nrofexons)+'exons'

#new\_fastaheader\_forsignalp = feature.id.split('.t')[0]+'\_'+genome+'\_contig'+contig+'\_'+str(gene\_start\_absolute)+'-'+str(gene\_end\_absolute)

new\_fastaheader\_id = 'sc'+contig+'|'+str(gene\_start\_absolute)+'-'+str(gene\_end\_absolute)+'|'+'d2m:'+str(gene\_start\_relative)+'bp|'+str(real\_orientation)+'|f?'+'|len:'+str(len(proteinseq))

new\_fastaheader\_description = str(mimp\_IR\_seq)+'^'+mimp\_IR\_loc+'\_'+str(nrofexons)+'\_exons'

print new\_fastaheader\_id, new\_fastaheader\_description

# SignalP only accepts short fastaheaders!!

#print proteinseq[:1]

if str(proteinseq[:1]) == 'M':

if len(proteinseq) > min\_prot\_len:

if len(proteinseq)<max\_prot\_len:

if int(gene\_start\_relative) < max\_d2m:

ORF\_list.append(SeqRecord(seq=ORF, id=new\_fastaheader\_id, description=new\_fastaheader\_description))

CDS\_list.append(SeqRecord(seq=CDS, id=new\_fastaheader\_id, description=new\_fastaheader\_description))

protein\_list.append(SeqRecord(seq=proteinseq, id=new\_fastaheader\_id, description=new\_fastaheader\_description))

genecounter+=1

print '\n// Putative genes near a mimp IR found in this genome: ' + str(genecounter)

print '// Wrote ORF (EXONintronEXON), mRNA coding sequences and protein sequences to datahandler\_3a/3b/3c.'

in\_handle.close()

SeqIO.write(ORF\_list, datahandler3a, 'fasta')

SeqIO.write(CDS\_list, datahandler3b, 'fasta')

SeqIO.write(protein\_list, datahandler3c, 'fasta')

def RunSignalP(datahandler3c, datahandler4, SignalPpath, SignalP\_threshold):

print '// Running SignalP 4.1...'

cline = SignalPpath+' -t euk -f summary -u %s %s > %s' % (SignalP\_threshold, datahandler3c, datahandler4)

os.system(cline)

def ParseSignalP(datahandler3a, datahandler3b, datahandler3c, datahandler4, datahandler5a, datahandler5b, datahandler5c, min\_prot\_len):

SP\_positives = []

with open(datahandler4, 'r+') as sp:

for line in sp:

if line.startswith('Name='):

sp\_name = line.split('Name=')[1].split('\tSP=')[0]

orfs = SeqIO.parse(datahandler3c, 'fasta')

for orf in orfs:

if orf.id == sp\_name:

sp\_present = line.split('\tSP=\'')[1].split('\'')[0] # SP: YES OR NO

orf.id += '|SP=' + sp\_present

orf.id += ';D='+line.split(' D=0')[1].split(' ')[0] # SP D-value

if sp\_present == 'YES':

cleavage\_site\_pos1 = line.split('Cleavage site between pos. ')[1].split(' and ')[0]

cleavage\_site\_pos2 = line.split('Cleavage site between pos. ')[1].split(' and ')[1].split(': ')[0]

if 'X' not in (orf.seq[:int(cleavage\_site\_pos2)-1]): #prevent 'NNN' translated to 'X' in signalpeptides to end up in the list

signalpeptideseq = str(orf.seq[:int(cleavage\_site\_pos2)-1])

#contig = orf.id.split('\_contig')[1].split('\_')[0]

orfie = SeqRecord(seq=signalpeptideseq.lower()+orf.seq[int(cleavage\_site\_pos2)-1:].upper(), id=orf.id, description=orf.description.split(' ')[1]+'\_signalpeptideseq='+signalpeptideseq)

SP\_positives.append(orfie)

print ' >'+orfie.id+' contains a signal peptide.'

sp.close()

print ' Total # of sequences matching the criteria: %i' % len(SP\_positives)

print ''

#Write all protein sequences that meet requirements (close to motif, longer than 30 aa and contains signal peptide) to datahandler5c:

SeqIO.write(SP\_positives, datahandler5c, 'fasta')

#Write all Open Reading Frame sequences (DNA) to datahandler5a

datahandler3a\_handle = SeqIO.parse(datahandler3a, "fasta")

SP\_positives\_ORFs=[]

for rec in datahandler3a\_handle:

#print ' ???? ', rec.id

for orfie in SP\_positives:

if rec.id == orfie.id.split('|SP=')[0]:

#print ' !!!! ', orfie.id.split('|SP=')[0]

new\_rec = SeqRecord(seq=rec.seq, id=orfie.id, description=orfie.description)

SP\_positives\_ORFs.append(new\_rec)

SeqIO.write(SP\_positives\_ORFs, datahandler5a, 'fasta')

#Write all Coding Sequences (mRNA) to datahandler5b

datahandler3b\_handle = SeqIO.parse(datahandler3b, "fasta")

SP\_positives\_CDS=[]

for rec in datahandler3b\_handle:

#print ' ???? ', rec.id

for orfie in SP\_positives:

if rec.id == orfie.id.split('|SP=')[0]:

#print ' !!!! ', orfie.id.split('|SP=')[0]

new\_rec = SeqRecord(seq=rec.seq, id=orfie.id, description=orfie.description)

SP\_positives\_CDS.append(new\_rec)

SeqIO.write(SP\_positives\_CDS, datahandler5b, 'fasta')

def ExtractOrfToFasta(proteinsfasta, uberinfile, puteff\_dnaseqs, genome, puteff\_logfile, combined\_puteff\_fasta, combined\_puteff\_logfile, filecounter, combined\_puteff\_logfile2, sc\_prefix):

#EXTRACT GENOMIC SEQUENSE:

open(puteff\_dnaseqs, 'wb').close() #clear genomic DNA sequence fastafile

dnaoutfile = open(puteff\_dnaseqs, 'a')

open(puteff\_logfile, 'wb').close()

logheader="genome\tputeff\_supercontig\tgenomic\_start\_pos\tgenomic\_end\_pos\tdist2mimp\torientation\tprotlength\tnrofexons\tD\_value\tmimp\_IR\_seq\tmimp\_IR\_pos\tsignalpeptideseq\tproteinseq\tgenomicsequence\n"

logheader2="genome\tnr\_of\_put\_eff\_Augustus\n"

puteff\_logfile\_writer = open(puteff\_logfile, 'a')

puteff\_logfile\_writer.write(logheader)

combined\_putefffile = open(combined\_puteff\_fasta, 'a')

combined\_puteff\_logfile\_writer = open(combined\_puteff\_logfile, 'a')

combined\_puteff\_logfile2\_writer = open(combined\_puteff\_logfile2, 'a')

if filecounter < 1: # During running of the first genome file, the log header should be added at the top.

combined\_puteff\_logfile\_writer.write(logheader)

combined\_puteff\_logfile2\_writer.write(logheader2)

proteinsfastafile = SeqIO.parse(proteinsfasta, 'fasta')

n=0

print '// Putative effectors found in genome: ' + genome

for seq\_record in proteinsfastafile:

n+=1

puteff\_supercontig = seq\_record.description.split('sc')[1].split('|')[0]

#MAKE SURE THE CONTIG IN THE ORIGINAL FASTA FILE IS LIKE THIS; contig\_1 (space no comma behind it)

genomic\_start\_pos = int(seq\_record.description.split('|')[1].split('-')[0])

genomic\_end\_pos = int(seq\_record.description.split('|')[1].split('-')[1].split('|')[0])

dist2mimp = seq\_record.description.split('d2m:')[1].split('bp')[0]

orientation = seq\_record.description.split('bp|')[1].split('|')[0]

protlength = seq\_record.description.split('|len:')[1].split('|')[0]

D\_value = seq\_record.description.split('D=.')[1].split(' ')[0]

mimp\_IR\_seq = seq\_record.description.split(' ')[1].split('^')[0]

mimp\_IR\_pos = seq\_record.description.split('^')[1].split('\_')[0]

nrofexons = seq\_record.description.split('^')[1].split('\_')[1].split('\_exons')[0]

signalpeptideseq = seq\_record.description.split('\_signalpeptideseq=')[1]

proteinseq = seq\_record.seq

all\_contigs = list(SeqIO.parse(uberinfile, "fasta", IUPAC.unambiguous\_dna))

for sc in all\_contigs:

sc\_id = sc.description.split(sc\_prefix)[1]

if "\_" in sc\_id:

sc\_id = sc\_id.split("\_")[0]

if sc\_id == puteff\_supercontig:

genomicsequence = sc.seq[genomic\_start\_pos-1:genomic\_end\_pos]

print ' contig\_'+str(puteff\_supercontig)+'\tposition '+str(genomic\_start\_pos)+'-'+str(genomic\_end\_pos)+'\t'+signalpeptideseq

if genomic\_start\_pos > genomic\_end\_pos:

genomicsequence = sc.seq[genomic\_end\_pos-1:genomic\_start\_pos]

putEff\_fastaentry = ">"+str(n).zfill(3)+'.'+signalpeptideseq+"\_"+genome+"\_d2m"+str(dist2mimp)+"\_len"+str(protlength)+"\n"+str(genomicsequence)+"\n\n"

dnaoutfile.write(putEff\_fastaentry)

puteff\_attributes = [genome, puteff\_supercontig, genomic\_start\_pos, genomic\_end\_pos, dist2mimp, orientation, protlength, nrofexons, D\_value, mimp\_IR\_seq, mimp\_IR\_pos, signalpeptideseq, proteinseq, genomicsequence]

putEff\_logentry = ('\t'.join(map(str,puteff\_attributes)))+'\n'

puteff\_logfile\_writer.write(putEff\_logentry)

#combined\_putefffile will collect all the output from the mimpsearch; this means there will be many redundant put effectors.

combined\_putefffile.write(putEff\_fastaentry)

combined\_puteff\_logfile\_writer.write(putEff\_logentry)

putEff\_logentry2 = genome+'\t'+str(n)+'\n'

combined\_puteff\_logfile2\_writer.write(putEff\_logentry2)

dnaoutfile.close() #collects inside genome out folder all DNA sequences of the putative effectors

puteff\_logfile\_writer.close() #writes a log for all puteff found in the current genome (inside genome out folder)

combined\_putefffile.close() #collects inside the out folder all DNA sequences of the putative effectors of all genomes that are being processed by the script.

combined\_puteff\_logfile\_writer.close() #writes a general log file with more details of the putative effectors identified

print '-'\*20

print "// Finished with genome of %s; wrote %i genomic DNA sequences of putEff ORFs to %s" % (genome, n, puteff\_dnaseqs)

def MainDef(genome, genomefastafile, directory, folder, combined\_puteff\_fasta, combined\_puteff\_logfile, filecounter, combined\_puteff\_logfile2, combined\_puteff\_dir):

# Add a string as an argument when running the Python script so to tell what is the supercontig prefix.

# ex: "python mimpfinder.py Supercontig\_"

sc\_prefix = sys.argv[3] #'contig\_'

AUGUSTUS\_command = sys.argv[4]+' '+sys.argv[5]

infilename, infileextension = os.path.splitext(genomefastafile)

infile = directory+folder+'/'+genomefastafile

outdirectory = combined\_puteff\_dir+infilename+'\_mimpfinder\_out/'

if not os.path.exists(outdirectory):

os.makedirs(outdirectory)

datahandler = outdirectory+infilename+'\_1\_mimpfinder\_downstreamregions.fasta'

datahandler2 = outdirectory+infilename+'\_2\_mimpfinder\_predictedgenes.gff'

datahandler3a = outdirectory+infilename+'\_3a\_mimpfinder\_ORFs.fasta'

datahandler3b = outdirectory+infilename+'\_3b\_mimpfinder\_CDS.fasta'

datahandler3c = outdirectory+infilename+'\_3c\_mimpfinder\_proteins.fasta'

datahandler4 = outdirectory+infilename+'\_4\_mimpfinder\_SignalP.summary\_out'

datahandler5a = outdirectory+infilename+'\_5a\_mimpfinder\_ORFs\_SP.fasta'

datahandler5b = outdirectory+infilename+'\_5b\_mimpfinder\_CDS\_SP.fasta'

datahandler5c = outdirectory+infilename+'\_5c\_mimpfinder\_proteins\_SP.fasta'

datahandler6 = outdirectory+infilename+'\_6\_mimpfinder\_ORFs\_SP.fasta'

puteff\_logfile = outdirectory+infilename+'\_7\_mimpfinder\_puteff\_logfile.txt'

motiefje = 'TT[TA]TTGC..CCCACTG..'

motiefje\_rc = '..CAGTGGG..GCAA[TA]AA'

#..CAGT[GA]G[GA]..GCAA[TAG]AA (Mara Bergemann, 2008)

orfs = []

distance = int(sys.argv[6]) # sequence downstream of motif used for ORF prediction

min\_prot\_len = int(sys.argv[7]) #in aa

max\_prot\_len = int(sys.argv[8])

max\_d2m = int(sys.argv[9]) # max distance between mimp TIR and start-codon

SignalPpath =str(sys.argv[10])

SignalP\_threshold = str(sys.argv[11])

MimpFinder(infile, sc\_prefix, motiefje, motiefje\_rc, datahandler, distance) #ir\_dict[i] = [m.start()+1, m.end(), seq\_record.seq[m.start():m.end()], seq\_record.seq[m.end():m.end()+distance]]

PredictGenes(datahandler, datahandler2, datahandler3a, datahandler3b, datahandler3c, genome, max\_d2m, AUGUSTUS\_command, min\_prot\_len, max\_prot\_len)

RunSignalP(datahandler3c, datahandler4, SignalPpath, SignalP\_threshold)

ParseSignalP(datahandler3a, datahandler3b, datahandler3c, datahandler4, datahandler5a, datahandler5b, datahandler5c, min\_prot\_len)

ExtractOrfToFasta(datahandler5c, infile, datahandler6, infilename, puteff\_logfile, combined\_puteff\_fasta, combined\_puteff\_logfile, filecounter, combined\_puteff\_logfile2, sc\_prefix)

if \_\_name\_\_ == "\_\_main\_\_":

######################

directory\_folder = sys.argv[1]

folder = sys.argv[1].split('/')[-1]

directory = directory\_folder.split(folder)[0]

output\_dir = sys.argv[2]

file\_extensions = (".fa", ".fasta", ".fas", "fna") # Specify the suffix of the genome files (.fa, .fasta, etc)

combined\_puteff\_dir = output\_dir+'/01.mimpfinder/'+folder+'\_AugustusOut/'

if not os.path.exists(combined\_puteff\_dir):

os.makedirs(combined\_puteff\_dir)

combined\_puteff\_fasta = combined\_puteff\_dir+'all\_putative\_effectors.fasta'

open(combined\_puteff\_fasta, 'w').close()

combined\_puteff\_logfile = combined\_puteff\_dir+'all\_putative\_effectors\_log.txt'

open(combined\_puteff\_logfile, 'w').close()

combined\_puteff\_logfile2 = combined\_puteff\_dir+'all\_putative\_effectors\_log2.txt'

open(combined\_puteff\_logfile2, 'w').close()

######################

filecounter=0

for genomefastafile in os.listdir(directory\_folder):

if genomefastafile.endswith(file\_extensions):

genome = genomefastafile.split('.fa')[0]

print "\n// executing mimpfinder\_AUGUSTUS script for "+genomefastafile

MainDef(genome, genomefastafile, directory, folder, combined\_puteff\_fasta, combined\_puteff\_logfile, filecounter, combined\_puteff\_logfile2, combined\_puteff\_dir)

filecounter+=1

else:

print '-'\*20

print '// THE END'

print "No more files with extension '%s' were found in directory '%s'" % (file\_extensions, (directory+folder))

print "Executed the script for %i files." % (filecounter)

print 'Total run time = %s' % (datetime.now()-startTime)

print '-'\*20

## Bijlage 3 - 02.cluster\_putefflists.py

#this script takes in the output of mimp-identification script "all\_putative\_effectors.fasta"

#It will take each put effector in this fastafile and blast it against itself. Then it takes these groups

#together and clusters them with single\_linkage. Each of the clusters is assessed for the longest

#sequence, and these longest sequences are written to the outfile (\_clustered.fasta)

from Bio import SeqIO

import os, sys

debug=0

def cluster\_homologous\_effectors(infile, E\_VALUE\_THRESH, PERC\_IDENTITY\_THRESH, LENGTH\_THRESH, blastdatabasedir, BLASTbindir, infiledir):

database\_store = blastdatabasedir+'/'+infile.split('/')[-1].split('.fa')[0]

cmnd = BLASTbindir+'/makeblastdb -dbtype nucl -in '+infile+' -out '+database\_store

print "---BLASTDB---\n", cmnd, os.system(cmnd), "---\n" #uncomment if you want to rebuild a blastdb #untag # if you want to build the BLAST database

blastoutfilename = infiledir+infile.split('/')[-1].split('.fa')[0]+'.vs.'+infile.split('/')[-1].split('.fa')[0]+'.blastout'

cmnd = BLASTbindir+"/blastn -outfmt '6 qseqid sseqid pident length mismatch gapopen qstart qend sstart send evalue bitscore qlen' -query "+infile+' -db '+database\_store+' -out '+blastoutfilename+' -evalue '+str(E\_VALUE\_THRESH)

# 0 1 2 3 4 5 6 7 8 9 10 11 12

effector2homologs = {}

if os.system(cmnd) == 0:

lines = open(blastoutfilename).readlines()

#'qseqid sseqid pident length mismatch gapopen qstart qend sstart send evalue bitscore qlen'

for line in lines:

tabs = line.strip().split('\t')

if float(tabs[2]) > PERC\_IDENTITY\_THRESH and (int(tabs[7]) - int(tabs[6]))/float(tabs[12]) > LENGTH\_THRESH: #with last check: from 358 to 354 put effectors with same setup

if effector2homologs.has\_key(tabs[0]):

effector2homologs[tabs[0]].add(tabs[1]) #add all BLAST-associated hits to this entry

else: effector2homologs[tabs[0]] = set([tabs[1]])

if effector2homologs.has\_key(tabs[1]):

effector2homologs[tabs[1]].add(tabs[0]) #the other way around; also add the BLAST query as an association to each BLAST hit.

else: effector2homologs[tabs[1]] = set([tabs[0]])

return single\_linkage(effector2homologs)

def single\_linkage(node\_partners):

clusters=[]

nodes=node\_partners.keys()

while len(nodes)>0:

if debug:

print 'new cluster, seed node:', nodes[0]

print '# keys in hash:', len(nodes)

cluster=update\_cluster(node\_partners, [nodes[0]], set([]))

clusters.append(cluster)

nodes=node\_partners.keys()

if debug:

print 'finished cluster:', cluster

print '# keys in hash:', len(nodes)

return clusters

def update\_cluster(node\_partners, todo, cluster=set([])):

if debug: print todo, cluster

new\_todo=set([])

for t in todo:

partners=node\_partners[t]

new\_todo=new\_todo.union(set(partners))

cluster.add(t)

del node\_partners[t]

new\_todo=new\_todo.difference(cluster)

if len(new\_todo)>0: return update\_cluster(node\_partners, new\_todo, cluster)

else: return cluster

if \_\_name\_\_ == "\_\_main\_\_":

### arguments being passed from pipeline script: ###

infile = sys.argv[1]

infile\_filename = infile.split('/')[-1]

infiledir = infile.split(infile\_filename)[0]

outfile = infile.split('.fa')[0]+'\_clustered.fasta'

blastdatabasedir = sys.argv[2]

leave\_put\_eff\_identifiers\_during\_clustering = sys.argv[3]

BLASTbindir = sys.argv[4]

###

outfilewriter = open(outfile, 'w')

E\_VALUE\_THRESH = 0.001

PERC\_IDENTITY\_THRESH = 60

LENGTH\_THRESH = 0.3

print '\n'

print "// Running clustering script on file %s" % infile

clusters = cluster\_homologous\_effectors(infile, E\_VALUE\_THRESH, PERC\_IDENTITY\_THRESH,LENGTH\_THRESH, blastdatabasedir, BLASTbindir, infiledir)

longest\_elements = []

handle = open(infile, "rU")

record\_dict = SeqIO.to\_dict(SeqIO.parse(handle, "fasta"))

handle.close()

for c in clusters:

print '// # of nodes in this cluster = ', str(len(c))

longest\_elem = [0, '']

for elem in c:

print elem

#elem\_len = int(elem.split('len')[1])

elem\_len = len(record\_dict[elem].seq)

######################################################################

### in case these pieces of information are going to be necessary: ###

#elem\_d2m = elem.split('\_len')[0].split('d2m')[1]

#elem\_isolate = elem.split('\_', 1)[1].split('\_d2m')[0]

#elem\_identifier = elem.split('.', 1)[0]

#elem\_idmethod = ''

#if len(str(elem\_identifier)) == 3:

#elem\_idmethod = 'augustus'

#elif len(str(elem\_identifier)) == 4:

#elem\_idmethod = 'metstop'

#print elem\_d2m, elem\_isolate, elem\_identifier, elem\_idmethod

######################################################################

if elem\_len > longest\_elem[0]:

longest\_elem[0] = elem\_len

longest\_elem[1] = elem

print '\t', elem\_len, '\t', elem

print '\t-------'

print '\tLongest element in cluster is: ', longest\_elem[1], '\n'

longest\_elements.append(record\_dict[longest\_elem[1]])

#SeqIO.write(longest\_elements, outfile, "fasta")

for longest\_element in longest\_elements:

if leave\_put\_eff\_identifiers\_during\_clustering == 'TRUE':

longest\_element.id = longest\_element.id #in case you're clustering a clustered file (that does not contain any description, but only a .id)

else:

longest\_element.id = longest\_element.description.split('.', 1)[1].split('\_', 1)[0] #split 1x after identifier (001.) and parse the rest to be the new name

longest\_element.description = ''

if "\x00" in longest\_element.seq:

print longest\_element.id

SeqIO.write(longest\_elements, outfile, "fasta")

outfilewriter.close()

print '-'\*30

print '// Found %i putative effectors in these genomes: ' % len(clusters)

for longest\_elem in longest\_elements:

print '\t'+longest\_elem.id

print '-'\*30

## Bijlage 4 - 03.local\_blast\_clustered\_putefflist\_to\_pres-abs\_table.py

from Bio.Blast.Applications import NcbiblastnCommandline, NcbitblastnCommandline

from Bio.Blast import NCBIXML

from Bio import SeqIO

import csv, os, re, sys, glob, datetime

def BuildBlastDB(genome\_folder, genome\_fastafile, blastdatabasedir, BLASTbindir):

database\_in = genome\_folder+'/'+genome\_fastafile

genome = genome\_fastafile.split('.fa')[0]

database\_store = blastdatabasedir + '/' + genome

cmnd = BLASTbindir+'/makeblastdb -dbtype nucl -in '+database\_in+' -out '+database\_store

if buildblastdb == 'yes':

print "// ---BUILDING BLAST DB FROM FASTA FILE---"

print cmnd, os.system(cmnd)

elif buildblastdb == 'no':

print "// No BLAST db will be built\n\n"

pass

elif buildblastdb != 'yes':

print "// No BLAST db will be built, use set buildblastdb to 'yes' to build a BLAST database from fastafiles."

return database\_store, genome

def BlastAndParse(blastdb, genome, queryfile, E\_VALUE\_THRESH, PERC\_IDENTITY\_THRESH, outputdir, BLAST\_task):

#blastoutput = outputdir+BLAST\_type+"\_presence\_absence.blastout"

blastoutput = outputdir+BLAST\_type+queryfile.split('/')[-1]+"\_vs"+genome+"\_presence\_absence.blastout"

open(blastoutput, 'wb').close() #clearfile

if BLAST\_type == 'BLASTN':

blastn\_cline = NcbiblastnCommandline(query=queryfile, db=blastdb, evalue=E\_VALUE\_THRESH, outfmt=5, out=blastoutput, dust='no', task=BLAST\_task)

print blastn\_cline

#stdout, stderr = blastn\_cline()

blastn\_cline()

else:

print "ERROR: NO BLAST TYPE HAS BEEN SPECIFIED (BLASTN OR TBLASTN)"

result\_handle = open(blastoutput)

blast\_records = NCBIXML.parse(result\_handle)

effector2scoresForThisGenome = {}

for blast\_record in blast\_records:

for alignment in blast\_record.alignments:

for hsp in alignment.hsps:

return\_perc\_id = int(100\*hsp.identities/blast\_record.query\_length)

#alignment\_length = abs(int(hsp.query\_end-hsp.query\_start))

#return\_perc\_id = int(100\*hsp.identities/alignment\_length)

########

return\_contig\_nr = "??"

subject\_name = alignment.title

if "ontig\_" in subject\_name:

return\_contig\_nr = subject\_name.split("ontig\_")[1]

if "\_" in return\_contig\_nr:

return\_contig\_nr = return\_contig\_nr.split("\_")[0]

if not effector2scoresForThisGenome.has\_key(blast\_record.query): effector2scoresForThisGenome[blast\_record.query]=[]

if (hsp.expect < E\_VALUE\_THRESH) and (return\_perc\_id >= PERC\_IDENTITY\_THRESH):

if output\_contigs in no: #'', 'no' or 'n'

effector2scoresForThisGenome[blast\_record.query].append(return\_perc\_id)

#print blast\_record.query, '\t', effector2scoresForThisGenome[blast\_record.query][0]

elif output\_contigs in yes:

effector2scoresForThisGenome[blast\_record.query].append(return\_contig\_nr+'('+str(hsp.sbjct\_start)+'-'+str(hsp.sbjct\_end)+')')

#else: #requirements are not met, so key with value '[]' should be popped out: """this doesn't work and shows no output whatsoever 26-3-2015"""

#effector2scoresForThisGenome.pop(blast\_record.query, None)

result\_handle.close()

return effector2scoresForThisGenome

if \_\_name\_\_ == "\_\_main\_\_":

### arguments being passed from pipeline script: ###

queryfile = sys.argv[1]

queryfilename = queryfile.split('/')[-1]

queryfolder = queryfile.split(queryfilename)[0]

genome\_folder = sys.argv[2]

genome\_folder\_b = genome\_folder.split('/')[-1]

genome\_folder\_a = genome\_folder.split(genome\_folder\_b)[0]

blastdatabasedir = sys.argv[3]

BLASTbindir = sys.argv[4]

outputdir = sys.argv[5]+'/03.blastn\_presence\_absence/'

if not os.path.exists(outputdir):

os.makedirs(outputdir)

###

date\_time\_now = datetime.datetime.now().strftime("%d-%m-%Y %H:%M")

queryfile = queryfolder+queryfilename

outfilename = outputdir+genome\_folder\_b+'.vs.'+queryfilename.split('.fa')[0]+'.txt'

E\_VALUE\_THRESH = 0.001

PERC\_IDENTITY\_THRESH = int(sys.argv[6])#30

BLAST\_type = 'BLASTN'

BLAST\_task = sys.argv[7]#'blastn' or megablast

buildblastdb = sys.argv[8]#'yes'

transpose = True

plot\_copynumber = False #define whether presence is '1' or copy nr (# of blast hits)

if buildblastdb == 'yes':

print "// Will start with building BlastDBs of each genome encountered.\n"

yes = set(['yes','y'])

no = set(['no','n', ''])

output\_contigs = 'n'

#hierarchicalclusteringquestion = raw\_input("Would you like to get hierarchical clustering output in stead of %id? (y/n) \n>")

hierarchicalclusteringquestion = 'y'

if hierarchicalclusteringquestion in yes:

hierarchicalclustering = True

outfilename = outfilename.split('.txt')[0]+'\_hierarch-clust.txt'

else:

hierarchicalclustering = False

output\_contigs = raw\_input("Would you like to get contig numbers in stead of %id? (y/n) \n>")

if output\_contigs in yes:

outfilename = outfilename.split('.txt')[0]+'\_contignrs.txt'

if transpose == True:

outfilename = outfilename.split('.txt')[0]+'\_transposed.txt'

query\_list = []

for record in SeqIO.parse(queryfile, 'fasta'):

query\_list.append(record.id)

###############################################################

#cancels all above outputfile naming statements:

outfilename = outputdir+'blastn\_presence\_absence.txt'

###############################################################

effector2genome2scores = {}

genomes = []

for genome\_fastafile in os.listdir(genome\_folder):

if genome\_fastafile.endswith((".fa", ".fasta", ".fas")):

blastdb, genome = BuildBlastDB(genome\_folder, genome\_fastafile, blastdatabasedir, BLASTbindir)

genomes.append(genome)

#return a dictionary containing all query fastaheaders (keys) with BLAST% as values:

effector2scoresForThisGenome = BlastAndParse(blastdb, genome, queryfile, E\_VALUE\_THRESH, PERC\_IDENTITY\_THRESH, outputdir, BLAST\_task)

#order these back to the way they were in the original query fastafile:

n=1

for effector in query\_list:

new\_record\_id = '{0:04}'.format(n)+'..'+effector #'..' to reduce mis-splitting

if not effector2genome2scores.has\_key(new\_record\_id): effector2genome2scores[new\_record\_id] = {}

#in dictionary effector2scoresForThisGenome, add empty values (for effectors that were not found with BLAST:

if not effector2scoresForThisGenome.has\_key(effector):

effector2scoresForThisGenome[effector] = []

effector2genome2scores[new\_record\_id][genome] = effector2scoresForThisGenome[effector]

n+=1

else:

print '-'\*20

print "// No more files with extension .fa, .fas, .fasta were found in directory '%s'" % (genome\_folder)

print '-'\*20

# Print to table:

open(outfilename, 'w').close() #clear file

outfile = open(outfilename, 'a')

genomes.sort()

header = ''

if hierarchicalclustering == False:

if output\_contigs in no:

header = BLAST\_type+' PERCENT IDENTITY\nDate: '+date\_time\_now+'\nE-value: '+str(E\_VALUE\_THRESH)+'\n'+'%ID\_threshold: '+str(PERC\_IDENTITY\_THRESH)+'%\n''query:'

elif output\_contigs in yes:

header = BLAST\_type+' CONTIG NR + SUBJECT LOCATION\nDate: '+date\_time\_now+'\nE-value: '+str(E\_VALUE\_THRESH)+'\n'+'%ID\_threshold: '+str(PERC\_IDENTITY\_THRESH)+'%\n''query:'

if transpose == True:

for effector in sorted(effector2genome2scores.keys()):

header += '\t'+effector

else:

for g in genomes:

header += '\t'+g

header+='\n'

print header

outfile.write(header)

if transpose == True:

for g in genomes:

out = g

for effector in sorted(effector2genome2scores.keys()):

genome2scores = effector2genome2scores[effector]

scores = genome2scores[g]

if hierarchicalclustering == True:

scorestr = 0

for s in scores:

if plot\_copynumber == True:

scorestr +=1

else:

scorestr =1

else:

scorestr = ''

for s in scores:

scorestr+=str(s)+';'

scorestr = scorestr[:-1] # laatste ; eraf

out += '\t'+str(scorestr)

print out

outfile.write(out+'\n')

else:

for effector in sorted(effector2genome2scores.keys()):

out = effector

genome2scores = effector2genome2scores[effector]

for g in genomes:

#print g

scores = genome2scores[g]

if hierarchicalclustering == True:

scorestr = 0

for s in scores:

scorestr=1

else:

scorestr = ''

for s in scores:

scorestr+=str(s)+';'

scorestr = scorestr[:-1] # laatste ; eraf

out += '\t'+str(scorestr)

print out

outfile.write(out+'\n')

outfile.close()

print '-'\*30

print "// Written data to file: ", outfilename

print '-'\*30

## Bijage 5 – 04.cluster\_and\_plot\_heatmap3.R

print('-------------------------------')

print("//Executing R script for clustering and plotting into a tree")

options(warn=-1) #Set this value back to 0 if you want to display Rscript warnings in the terminal.

args <- commandArgs(trailingOnly = TRUE)

suppressPackageStartupMessages(library(dendextend))

library("gplots")

library("ctc")

library("extrafont")

library("ade4")

currentdir=getwd()

heatmap3path=args[1]

source(heatmap3path)

infile=args[2]

outputdir=args[3]

hierclust\_plot = "hierclust\_plot.pdf"

distance\_matrix\_rows = args[4]

clustering\_method\_rows = args[5]

distance\_matrix\_cols = args[6]

clustering\_method\_cols = args[7]

#1 = Jaccard index (1901) S3 coefficient of Gower & Legendre s1 = a / (a+b+c)

#2 = Simple matching coefficient of Sokal & Michener (1958) S4 coefficient of Gower & Legendre s2 = (a+d) / (a+b+c+d)

#3 = Sokal & Sneath(1963) S5 coefficient of Gower & Legendre s3 = a / (a + 2(b + c))

#4 = Rogers & Tanimoto (1960) S6 coefficient of Gower & Legendre s4 = (a + d) / (a + 2(b + c) +d)

#5 = Dice (1945) or Sorensen (1948) S7 coefficient of Gower & Legendre s5 = 2a / (2a + b + c)

#6 = Hamann coefficient S9 index of Gower & Legendre (1986) s6 = (a - (b + c) + d) / (a + b + c + d)

#7 = Ochiai (1957) S12 coefficient of Gower & Legendre s7 = a / sqrt((a + b)(a + c))

#8 = Sokal & Sneath (1963) S13 coefficient of Gower & Legendre s8 = ad / sqrt((a + b)(a + c)(d + b)(d + c))

#9 = Phi of Pearson S14 coefficient of Gower & Legendre s9 = (ad - bc) / sqrt((a + b)(a + c)(d + b)(d + c))

#10 = S2 coefficient of Gower & Legendre S10 = a / (a + b + c + d)

setwd(outputdir) #this folder should exist!

outfile <- "blastn\_presence\_absence\_reordered.txt"

d <- read.table(infile, sep = "\t", header = TRUE)

row.names(d) <- d[,1] #rename rows to values in first collumn

d[,1] <- NULL #remove the first collumn

title <- paste("Presence of candidate sequences in", nrow(d), "isolates and", ncol(d), "ORFs: \n", distance\_matrix\_rows, clustering\_method\_rows, distance\_matrix\_cols, clustering\_method\_cols)

data <- as.matrix(d)

#distance = Dist(data, method = 'pearson')

distance = dist.binary(data, method=distance\_matrix\_rows, diag = FALSE, upper = FALSE)

cluster = hclust(distance, method=clustering\_method\_rows)

dendrogram = as.dendrogram(cluster)

Rowv = rowMeans(data, na.rm = T)

dendrogram = reorder(dendrogram, Rowv) %>% set("branches\_lwd", 2) #%>% set("branches\_k\_color", k = 10)

#dendrogram <- color\_branches(dendrogram, k = 9, col = c("black", "forestgreen", "forestgreen", "blue", "blue", "darkred", "purple", "orange", "limegreen"))

###

#coldistance = Dist(t(data), method ='pearson')

#print(cor(data))

coldistance = dist.binary(t(data), method=distance\_matrix\_cols, diag = FALSE, upper = FALSE)

colcluster = hclust(coldistance, method=clustering\_method\_cols)

coldendrogram = as.dendrogram(colcluster) %>% ladderize(right = TRUE) %>% set("branches\_lwd", 2)

Colv = colMeans(data, na.rm = T)

coldendrogram = reorder(coldendrogram, Colv)

#coldendrogram <- color\_branches(coldendrogram, k = 5, col = c("darkblue", "darkred", "darkred", "forestgreen", "black"))

## Re-order the original data using the computed dendrogram

rowInd = rev(order.dendrogram(dendrogram))

colInd = order.dendrogram(coldendrogram)

data\_ordered <- data[rowInd, colInd]

#and write to a TXT file:

write.table(data\_ordered, outfile, quote=F, sep="\t",row.names=T, col.names=T)

#export tree to newick format

write(hc2Newick(cluster),file="cluster\_rows.newick")

write(hc2Newick(colcluster),file="cluster\_cols.newick")

#for the data matrix to be plotted ('data'), check if 'SIX' in name. In case this is true, plot that name.

column\_annotation = matrix("#dbdbdb", ncol=2, nrow = ncol(data))

sixgenecodes <- list('SIX','MAPYSM','MAPYGIV','MKVALV','MQPLRI','MKLSAV','MLVSPI','MAPYSM', 'MKLLWL', 'MFSKAI', 'MTRFHL', 'MHTEYLF', 'MLFKIAW', 'MRFLLLIA', 'MNLKALVV', 'MRFEYI', 'MKLALIA', 'MKYLYLL', 'MDRTHRG', 'MFVSPKA', 'MNLKALVV')

sixgene\_list = list()

for (name in sixgenecodes){

sixgene\_list[length(sixgene\_list)+1] <- list(grep(name, colnames(data)))

}

for (sixgene\_integer in sixgene\_list) {

column\_annotation[sixgene\_integer,1] = "red"

}

LScodes <- list('enz\_')

LS\_list = list()

for (name in LScodes){

LS\_list <- list(grep(name, colnames(data)))

}

for (LS\_integer in LS\_list) {

column\_annotation[LS\_integer,2] = "black"

}

colnames(column\_annotation) <- c("SIX genes", "Secreted enzymes")

#rownames(NA/NaN/Inf in foreign function call (arg 11)

rownames(column\_annotation) <- colnames(data)

pdf(file=hierclust\_plot, width=12, height=16, pointsize = 8) # default pointsize = 16 , family="Arial"

heatmap.3(data,

Rowv=dendrogram,

Colv=coldendrogram,

dendrogram="both",

#col=colorpanel(10, low="#D0D8EE",high="#233F88"),

col=colorpanel(10, low="#dbdbdb",high="#233F88"),

key=FALSE,

density.info="none",

trace="none",

labCol=colnames(data),

#cexCol=4,

sepcolor="#000000",

#rowsep=c(0, 9, 12,22, 28, 31, 32, 33, 49, 52, 54, 56, 57, 58, 59, 60, 61),

#rowsep=c(0, 4,7,16,21,25,26,37,38,43,46,47,48,49,50,51,53,56,58,59,60,61),

#colsep=1:ncol(data),

#colsep=c(18,28, 45,59, 77),

main=title,

cex.main = 1,

margins=c(26,16), #default 9,10

#RowSideColors= myClusterSideBar

ColSideColors = column\_annotation

)

dev.off()

print('Finished R script for clustering and plotting.')

## Bijlage 6 – FoEC.py

import os, datetime

starttime\_unformatted = datetime.datetime.now()

starttime = datetime.datetime.now().strftime("%y.%m.%d\_%Hh%Mm%S")

from optparse import OptionParser

##########################################################

####[EDIT THESE VARIABLES BEFORE RUNNING THE PIPELINE]####

##########################################################

#paths

blastdatabasedir = '/Users/Peter/Documents/Sequences/Fo\_genomes/blastdbs'

contigprefix = 'contig\_' # default: contig\_

AUGUSTUS\_path = '/Users/Peter/Programming/augustus-3.1/bin/augustus'

AUGUSTUS\_CONFIG\_path = '/Users/Peter/Programming/augustus-3.1/config'

BLASTbindir = '/usr/local/ncbi/blast/bin'

SignalPpath = '/Users/Peter/Programming/signalp/signalp-4.1/signalp'

#mimp-search variables

distance\_MetStop = '2500' # sequence downstream of motif used for ORF prediction

distance\_Augustus = '5000' # sequence downstream of motif used for ORF prediction

min\_prot\_len = '25' # in aa

max\_prot\_len = '600' # in aa

max\_d2m = '2000' # max distance between mimp TIR and start-codon

SignalP\_threshold = '0.550'

#blast variables

PERC\_IDENTITY\_THRESH = '30'

BLAST\_task = 'blastn' # or megablast

buildblastdb = 'yes' # should a new blast db be built for the genome files encountered? (recommended for first time this script is run on a set of genomes)

#clustering variables:

distance\_matrix\_rows = '1'

clustering\_method\_rows = 'average'

distance\_matrix\_cols = '1'

clustering\_method\_cols = 'average'

# Clustering methods that can be used for the hierarchical clustering:

#1 = Jaccard index (1901) S3 coefficient of Gower & Legendre s1 = a / (a+b+c)

#2 = Simple matching coefficient of Sokal & Michener (1958) S4 coefficient of Gower & Legendre s2 = (a+d) / (a+b+c+d)

#3 = Sokal & Sneath(1963) S5 coefficient of Gower & Legendre s3 = a / (a + 2(b + c))

#4 = Rogers & Tanimoto (1960) S6 coefficient of Gower & Legendre s4 = (a + d) / (a + 2(b + c) +d)

#5 = Dice (1945) or Sorensen (1948) S7 coefficient of Gower & Legendre s5 = 2a / (2a + b + c)

#6 = Hamann coefficient S9 index of Gower & Legendre (1986) s6 = (a - (b + c) + d) / (a + b + c + d)

#7 = Ochiai (1957) S12 coefficient of Gower & Legendre s7 = a / sqrt((a + b)(a + c))

#8 = Sokal & Sneath (1963) S13 coefficient of Gower & Legendre s8 = ad / sqrt((a + b)(a + c)(d + b)(d + c))

#9 = Phi of Pearson S14 coefficient of Gower & Legendre s9 = (ad - bc) / sqrt((a + b)(a + c)(d + b)(d + c))

#10 = S2 coefficient of Gower & Legendre S10 = a / (a + b + c + d)

#######################################################################

scriptdir = os.path.dirname(os.path.realpath(\_\_file\_\_))

default\_outputdir = scriptdir+'/output/output\_'+starttime

scriptname = os.path.basename(\_\_file\_\_)

usage = '\n'+'-'\*20+'\npython FoEC.py -i [infolder] <options>\n'+'-'\*20+'\n\

This script will take a folder with genome fasta files, find mimps and mimp terminal inverted repeats \

and try to identify candidate effectors. These will be clustered into families and then BLASTed against \

each of the genomes to identify presence (1) or absence (0). These binary patterns will be hierarchically \

clustered in R to produce a clustering figure using "heatmap.3.R". \n-Peter van Dam (Oct 2016)'

parser = OptionParser(usage=usage)

parser.add\_option("-i", "--in", dest="infolder", help="provide a folder with genome fasta files. N.B. provide an ABSOLUTE path!")

parser.add\_option("-o", "--out", dest="outfolder", help="output folder; default is: "+default\_outputdir)

parser.add\_option("-e", "--effector\_fasta", dest="effector\_fasta", help="Skip the effector prediction pipeline and go straight to blasting and clustering using a self-supplied effector list in fasta format.")

parser.add\_option("-m", "--met\_stop\_prediction\_only", dest="met\_stop\_prediction\_only", action="store\_true", default=False,

help="Do not use Augustus gene prediction, only identify ORFs by looking for first occurrence of a Methionine (met) to first Stop codon (stop).")

parser.add\_option("-u", "--use\_nonredundant\_effectors", dest="use\_nonredundant\_effectors", action="store\_true", default=False,

help="In case multiple ORF prediction methods are used (NOT using -m), the outputs from these methods can be clustered first, then concatenated, then clustered (use this -u option).\

Default behaviour is not to use this option, meaning taking the output from both methods, concatenate them and then cluster them into gene families.")

(options, args) = parser.parse\_args()

if options.infolder == None:

print '\n[ERROR!]\nPlease specify a folder containing Fusarium oxysporum genomes: "-i [infolder]"'

print 'Use "-h" to see additional options.\n'

quit()

else:

genomedir = options.infolder

if " " in genomedir:

print '\n[ERROR!]\nThis input folder contains a space, which may lead to problems: '+genomedir+'\nExiting..'

genomefoldername = os.path.basename(os.path.normpath(genomedir))

if options.outfolder == None:

outputdir = default\_outputdir

else:

outputdir = options.outfolder

if options.effector\_fasta == None:

####[A. Identify mimp-terminal inverted repeats and find ORFs with a Signal Peptide within a downstream region]####

genomedirs = genomedir.split(',')

for genomedir in genomedirs:

if not os.path.exists(genomedir):

print '\n[ERROR!]\nThis input folder could not be found: '+genomedir+'\nExiting..'

quit()

cline1a = ' '.join(['python', scriptdir+'/01a.mimpfinder\_combine\_to\_putefflist\_MetStop.py', genomedir, outputdir, contigprefix, distance\_MetStop, min\_prot\_len, max\_prot\_len, max\_d2m, SignalPpath, SignalP\_threshold])

print cline1a, os.system(cline1a)

if options.met\_stop\_prediction\_only:

cline1b='No Augustus gene prediction will be executed.'

print cline1b

else:

cline1b = ' '.join(['python', scriptdir+'/01b.mimpfinder\_combine\_to\_putefflist\_AUGUSTUS.py', genomedir, outputdir, contigprefix, AUGUSTUS\_path, '--AUGUSTUS\_CONFIG\_PATH='+AUGUSTUS\_CONFIG\_path, distance\_Augustus, min\_prot\_len, max\_prot\_len, max\_d2m, SignalPpath, SignalP\_threshold])

print cline1b, os.system(cline1b)

####[B. Reduce redundancy in each putative effector output file:]####

all\_putative\_effectors\_MetStop = outputdir+'/01.mimpfinder/'+genomefoldername+'\_MetStopOut/all\_putative\_effectors.fasta'

all\_putative\_effectors\_Augustus = outputdir+'/01.mimpfinder/'+genomefoldername+'\_AugustusOut/all\_putative\_effectors.fasta'

leave\_put\_eff\_identifiers\_during\_clustering = 'TRUE'

cline2a = ' '.join(['python', scriptdir+'/02.cluster\_putefflists.py', all\_putative\_effectors\_MetStop, blastdatabasedir, leave\_put\_eff\_identifiers\_during\_clustering, BLASTbindir])

print cline2a, os.system(cline2a)

if options.met\_stop\_prediction\_only:

cline2b='No clustering of Augustus effectors necessary.'

print cline2b

else:

cline2b = ' '.join(['python', scriptdir+'/02.cluster\_putefflists.py', all\_putative\_effectors\_Augustus, blastdatabasedir, leave\_put\_eff\_identifiers\_during\_clustering, BLASTbindir])

print cline2b, os.system(cline2b)

# now, the outputs of both metstop and augustus are clustered.

####[C. Concatenate and cluster output from MetStop and Augustus (either using raw output or clustered output):]####

all\_putative\_effectors\_MetStop\_clustered = all\_putative\_effectors\_MetStop.split('.fa')[0]+'\_clustered.fasta'

all\_putative\_effectors\_Augustus\_clustered = all\_putative\_effectors\_Augustus.split('.fa')[0]+'\_clustered.fasta'

if not os.path.exists(outputdir+'/02.cluster\_putative\_effectors/'):

os.makedirs(outputdir+'/02.cluster\_putative\_effectors/')

all\_putative\_effectors\_concatenated = outputdir+'/02.cluster\_putative\_effectors/all\_putative\_effectors\_concatenated.fasta'

if options.met\_stop\_prediction\_only:

clinecat = 'cat '+all\_putative\_effectors\_MetStop+' > '+all\_putative\_effectors\_concatenated

else:

if options.use\_nonredundant\_effectors == False:

clinecat = 'cat '+all\_putative\_effectors\_MetStop+' '+all\_putative\_effectors\_Augustus+' > '+all\_putative\_effectors\_concatenated

else:

clinecat = 'cat '+all\_putative\_effectors\_MetStop\_clustered+' '+all\_putative\_effectors\_Augustus\_clustered+' > '+all\_putative\_effectors\_concatenated

print clinecat, os.system(clinecat)

####[D. Cluster fasta file containing all concatenated effectors identified so that each effector occurs once in the list:]####

leave\_put\_eff\_identifiers\_during\_clustering = 'FALSE'

cline2c = ' '.join(['python', scriptdir+'/02.cluster\_putefflists.py', all\_putative\_effectors\_concatenated, blastdatabasedir, leave\_put\_eff\_identifiers\_during\_clustering, BLASTbindir])

print cline2c, os.system(cline2c)

all\_putative\_effectors\_concatenated\_clustered = all\_putative\_effectors\_concatenated.split('.fa')[0]+'\_clustered.fasta'

else: #an own list of putative effectors was supplied to be blasted and clustered:

if options.effector\_fasta.endswith('.fa') or options.effector\_fasta.endswith('.fasta') or options.effector\_fasta.endswith('.fas'):

all\_putative\_effectors\_concatenated\_clustered = options.effector\_fasta

####[E. BLASTN for presence/absence scoring:]####

cline3 = ' '.join(['python', scriptdir+'/03.local\_blast\_clustered\_putefflist\_to\_pres-abs\_table.py', all\_putative\_effectors\_concatenated\_clustered, genomedir, blastdatabasedir, BLASTbindir, outputdir, PERC\_IDENTITY\_THRESH, BLAST\_task, buildblastdb])

print cline3, os.system(cline3)

cline3\_output = outputdir+'/03.blastn\_presence\_absence/blastn\_presence\_absence.txt'

####[F. Use R script for hierarchical clustering:]####

cline4\_outputdir = outputdir+'/04.cluster\_and\_plot/'

if not os.path.exists(cline4\_outputdir):

os.makedirs(cline4\_outputdir)

cline4 = ' '.join(['Rscript', scriptdir+'/04.cluster\_and\_plot\_heatmap3.R', scriptdir+'/heatmap.3.R', cline3\_output, cline4\_outputdir, distance\_matrix\_rows, clustering\_method\_rows, distance\_matrix\_cols, clustering\_method\_cols])

print cline4, os.system(cline4)

log\_txt = open(outputdir+"/log\_txt.txt", "w")

if options.effector\_fasta != None: # when providing a set of effectors to be BLASTed and clustered:

write\_this = "==[ Parameters that were set in FoEC.py: ]==\n"

write\_this += "PERC\_IDENTITY\_THRESH:"+str(PERC\_IDENTITY\_THRESH)+"\n"

write\_this += "BLAST\_task:"+str(BLAST\_task)+"\n"

write\_this += "buildblastdb:"+str(buildblastdb)+"\n"

write\_this += "distance\_matrix\_rows:"+str(distance\_matrix\_rows)+"\n"

write\_this += "clustering\_method\_rows:"+str(clustering\_method\_rows)+"\n"

write\_this += "distance\_matrix\_cols:"+str(distance\_matrix\_cols)+"\n"

write\_this += "clustering\_method\_cols:"+str(clustering\_method\_cols)+"\n\n"

write\_this += "==[ Commands that were executed: ]==\n"

write\_this += "cline3\n"+str(cline3)+"\n\n"

write\_this += "cline4\n"+str(cline4)+"\n\n"

else:

write\_this = "==[ Parameters that were set in FoEC.py: ]==\n"

write\_this += "distance\_MetStop:"+str(distance\_MetStop)+"\n"

write\_this += "distance\_Augustus:"+str(distance\_Augustus)+"\n"

write\_this += "min\_prot\_len:"+str(min\_prot\_len)+"\n"

write\_this += "max\_prot\_len:"+str(max\_prot\_len)+"\n"

write\_this += "max\_d2m:"+str(max\_d2m)+"\n"

write\_this += "SignalP\_threshold:"+str(SignalP\_threshold)+"\n"

write\_this += "PERC\_IDENTITY\_THRESH:"+str(PERC\_IDENTITY\_THRESH)+"\n"

write\_this += "BLAST\_task:"+str(BLAST\_task)+"\n"

write\_this += "buildblastdb:"+str(buildblastdb)+"\n"

write\_this += "distance\_matrix\_rows:"+str(distance\_matrix\_rows)+"\n"

write\_this += "clustering\_method\_rows:"+str(clustering\_method\_rows)+"\n"

write\_this += "distance\_matrix\_cols:"+str(distance\_matrix\_cols)+"\n"

write\_this += "clustering\_method\_cols:"+str(clustering\_method\_cols)+"\n\n"

write\_this += "==[ Commands that were executed: ]==\n"

write\_this += "cline1a\n"+str(cline1a)+"\n\n"

write\_this += "cline1b\n"+str(cline1b)+"\n\n"

write\_this += "cline2a\n"+str(cline2a)+"\n\n"

write\_this += "cline2b\n"+str(cline2b)+"\n\n"

write\_this += "clinecat\n"+str(clinecat)+"\n\n"

write\_this += "cline2c\n"+str(cline2c)+"\n\n"

write\_this += "cline3\n"+str(cline3)+"\n\n"

write\_this += "cline4\n"+str(cline4)+"\n\n"

copy\_cline = "cp "+scriptdir+"/"+scriptname+" "+outputdir+"/"+scriptname

os.system(copy\_cline)

write\_this += "==[ A copy of the current version of FoEC.py script has also been saved to: ]==\n"

write\_this += outputdir+'/'+scriptname

log\_txt.write(write\_this)

log\_txt.close()

print '-----------------------'

print '\n\nStarttime:', starttime

print 'Endtime', datetime.datetime.now().strftime("%y.%m.%d\_%Hh%Mm%S")

print '-----------------------'

print 'Total time used for whole script:', (datetime.datetime.now()-starttime\_unformatted)

print '-----------------------'

## Bijlage 7 – MimpFinder.py

import re, configparser

from os import path

from Bio import SeqIO

from Bio.Alphabet import IUPAC

def FindSequence(infile):

'''

picks the sequence from infile\_dc. Than it makes a new prefix (id) for in the fastafile that is made from this sript.

parameters: infile\_dc: The file that is given in the configfile.

return: sequence and the new id.

'''

full\_id\_sequence\_list = []

for seq\_record in SeqIO.parse(infile,'fasta', IUPAC.ambiguous\_dna):

id\_seq\_list = []

id\_seq\_list.append(seq\_record.description)

id\_seq\_list.append(seq\_record.seq)

full\_id\_sequence\_list.append(id\_seq\_list)

return full\_id\_sequence\_list

def MimpFinder(motive, sequence\_list, length):

'''Finds the mimps in the sequence. When it finds a mimp it makes a flank of the length given in the config file.

than the flank is put into a list with the header.

Parameters:

‘sequence’: a string, variable that contains only the sequence of the organism, without header

‘new\_id’: a string, the id that belongs to the sequence

'''

header\_flank\_list = []

tir\_start\_stop\_list = []

for id\_seq in sequence\_list:

for match in re.finditer(motive, str(id\_seq[1])):

start\_stop\_mimp = str(match.start() + 1)+ '-' + str(match.end())

eind\_flank = int(match.end() + length)

if eind\_flank > len(id\_seq[1]):

eind\_flank = len(id\_seq[1])

start\_flank = int(match.end())

start\_stop\_flank = str(start\_flank + 1) + '-' + str(eind\_flank)

flank = id\_seq[1][start\_flank:eind\_flank]

header\_flank\_list.append(str('>' + id\_seq[0] + '|mimp\_downstreamregion:' + start\_stop\_flank +

'\_(strand:+)\_' + match.\_\_getitem\_\_(0) + '^' + start\_stop\_mimp + '\n' + flank + '\n'))

tir\_start\_stop\_list.append([match.start() + 1, match.end()])

return header\_flank\_list, tir\_start\_stop\_list

def MimpFinder\_rc(motive, sequence\_list, length):

'''Finds the mimps in the sequence. When it finds a mimp it makes a flank of the length given in the config file.

than the flank is put into a list with the header.

Parameters:

‘sequence’: a string, variable that contains only the sequence of the organism, without header

‘new\_id’: a string, the id that belongs to the sequence

'''

header\_flank\_list = []

tir\_start\_stop\_list\_rc = []

for id\_seq in sequence\_list:

for match in re.finditer(motive, str(id\_seq[1])):

start\_stop\_mimp = str(match.start() + 1) + '-' + str(match.end())

eind\_flank = int(match.start())

start\_flank = int(match.start() - length)

if start\_flank < 0:

start\_flank = 0

start\_stop\_flank = str(start\_flank) + '-' + str(eind\_flank)

flank = id\_seq[1][start\_flank:eind\_flank]

rev\_match = reverse\_complement(match.\_\_getitem\_\_(0))

header\_flank\_list.append(str('>' + id\_seq[0] + '|mimp\_upstreamregion:' + start\_stop\_flank +

'\_(strand:-,rev.transc.)\_' + rev\_match + '^' + start\_stop\_mimp + '\n' + flank + '\n'))

tir\_start\_stop\_list\_rc.append([match.start() + 1, match.end()])

return header\_flank\_list, tir\_start\_stop\_list\_rc

def TIR\_finder(sequence\_list, tir\_start\_stop\_list, motive\_rc):

#contig in de header en tir compleet maken. inverted repeat vinden

tir\_list = []

for id\_seq in sequence\_list:

for start\_stop in tir\_start\_stop\_list:

tir = id\_seq[1][start\_stop[0]-400:start\_stop[0]]

for tir\_match in re.finditer(motive\_rc, str(tir)):

start = start\_stop[0]-tir\_match.start()

stop = start\_stop[1]

tir\_list.append(str('>tir\_location:' + str(start) + '-' + str(stop) +

'\n' + id\_seq[1][start:stop]+'\n'))

return tir\_list

def TIR\_finder\_rc(sequence\_list, tir\_start\_stop\_list, motive):

tir\_list\_rc = []

for id\_seq in sequence\_list:

for start\_stop in tir\_start\_stop\_list:

tir = id\_seq[1][start\_stop[0]:start\_stop[0]+400]

for tir\_match in re.finditer(motive, str(tir)):

start = start\_stop[0]

stop = start\_stop[0]+tir\_match.end()

tir\_list\_rc.append(str('>tir\_location:' + str(start) + '-' + str(stop) +

'\n' + id\_seq[1][start:stop] + '\n'))

return tir\_list\_rc

def reverse\_complement(dna):

'''makes a reverse complement version sequence

Parameters:

‘dna’: a string, the DNA sequence that needs to be in reverse complement

'''

complement = {'A':'T','C':'G','G':'C','T':'A'}

return ''.join([complement[base] for base in dna[::-1]])

def WriteToFile(header\_flank\_lijst, outputfile):

"""imput list of headers with flanks

function writes each element in the list to a file.

Parameters:

‘header\_flank\_lijst’: a string, the full header including start/stop flank, mimp match, start/stop mimp

and the sequence of the flank that has to be written to a file.

"""

text\_output = str(outputfile).split('/')[-1]

for header\_flank in header\_flank\_lijst:

with open(outputfile, 'a') as file:

file.write(header\_flank)

print(header\_flank.split('\n')[0] + ' is found and put into ' + text\_output)

def main():

config = configparser.ConfigParser()

config.read(snakemake.input.a) # Input a in snakemake file

# reads the config file and put it into variables

#file = config['Parameters']['file']

file = snakemake.input.b

print(file)

"""

if path.exists(subfile + ".fasta") == True:

file = subfile + "fasta"

elif path.exists(subfile + ".fa") == True:

file = subfile + "fa"

elif path.exists(subfile + ".fna") == True:

file = subfile + "fna"""

motive = config['Mimpfinder']['motive']

motive\_rc = config['Mimpfinder']['motive\_rc']

length = int(config['Mimpfinder']['length'])

outputfile = snakemake.output.a # Output a in snakemake file.

tir\_file = snakemake.output.b

# makes sequence list

sequence\_list = FindSequence(file)

#Finds flanks

header\_flank\_list, tir\_start\_stop\_list = MimpFinder(motive, sequence\_list, length)

header\_flank\_list\_rc, tir\_\_start\_stop\_list\_rc = MimpFinder\_rc(motive\_rc, sequence\_list, length)

#Writes flanks to file

if len(header\_flank\_list) != 0 or len(header\_flank\_list\_rc) != 0:

WriteToFile(header\_flank\_list, outputfile)

WriteToFile(header\_flank\_list\_rc, outputfile)

else:

print("\nNo MIMP's motives have been found.\n\nThe program will stop...\n")

exit()

#Finds complete MIMP's. If not, it writes 'No complete MIMP's have been found...'

tir\_list = TIR\_finder(sequence\_list, tir\_start\_stop\_list, motive\_rc)

if len(tir\_list) > 0:

WriteToFile(tir\_list, tir\_file)

else:

with open(snakemake.output.b, "a") as file:

file.write("No complete MIMP's have been found in the normal flank...\n")

file.close()

# Finds complete MIMP's. If not, it writes 'No complete MIMP's have been found...'

tir\_list\_rc = TIR\_finder\_rc(sequence\_list, tir\_\_start\_stop\_list\_rc, motive)

if len(tir\_list\_rc) > 0:

WriteToFile(tir\_list\_rc, tir\_file)

else:

with open(snakemake.output.b, "a") as file:

file.write("No complete MIMP's have been found in the rc flank...\n")

file.close()

main()

## Bijlage 8 – MetStop.py

import os, re, sys

import configparser

import textwrap

from Bio import SeqIO

from Bio.Seq import Seq

from Bio.SeqRecord import SeqRecord

from Bio.Alphabet import IUPAC

class MetStop:

"""

The class MetStop uses the content in the attribute 'file', a file containing DNA sequence(s) from the MIMP Finder module.

Firstly, all three frames are generated.

Secondly, the open reading frames (ORF) are found with sequences larger than 23 aminoacids and shorter than 300 aminoacids.

Thirdly, SignalP-5.0 is used to find possible signalpeptide sequences in the ORFs, this results in putative effectors.

At last, useful headers are generated for each putative effector and the sequences are written down in a FASTA-file.

"""

def \_\_init\_\_(self, file):

self.\_\_file = file

self.datahandler\_list = []

for seq\_record in SeqIO.parse(self.get\_file(), "fasta", IUPAC.ambiguous\_dna):

self.datahandler\_list.append(SeqRecord(

seq=seq\_record.seq,

id=seq\_record.id,

description=""))

self.parser()

self.three\_frame\_translation()

self.orf\_finder()

self.orf\_writer()

if self.signalversion == "4":

self.signalp4()

os.system("mv results.txt " + snakemake.output.b)

os.system("rm ORFS.fasta")

else:

self.signalp5()

self.signalp\_writer()

print("Warning: deleting files: MetStop\_mature.fasta, ORFS.fasta and results.txt")

os.system("rm MetStop\_mature.fasta ORFS.fasta results.txt")

os.system("mv MetStop\_summary.signalp5 " + snakemake.output.b)

def get\_file(self):

"""

Returns self.\_\_file

:return: a string, returning the file name with DNA sequences from the MIMP Finder module

"""

return self.\_\_file

def set\_file(self, n\_file):

"""

It changes self.\_\_file to n\_file.

:param n\_file: a string, containing the new file name

:return: none

"""

self.\_\_file = n\_file

return

def parser(self):

"""

The function parses the config file and returns the contents.

:return: a string containing the file name

"""

config = configparser.ConfigParser()

config.read("config.ini")

self.directory = config['Parameters']['name\_dir']

self.probability = config['MetStop']['probability']

self.min\_protein\_length = int(config['MetStop']['min\_protein\_length'])

self.max\_protein\_length = int(config['MetStop']['max\_protein\_length'])

self.signalversion = config["Parameters"]["signalp\_version"]

self.signalp\_path = config["Parameters"]["signalp\_path"]

return

def three\_frame\_translation(self):

"""

The function finds every possible frames using three frame translation method.

The sequences are not translated, yet.

:return: none

"""

self.translated\_id = []

self.nuc = []

for x in range(len(self.datahandler\_list)):

header = self.datahandler\_list[x].id

nucleotide\_string = self.datahandler\_list[x].seq

while len(nucleotide\_string) % 3 != 0:

nucleotide\_string = nucleotide\_string + "N"

for frame in range(0,4):

if header.find("strand:+") and frame != 0:

self.translated\_id.append(">" + header + "|frame:" + "+" + str(frame))

elif header.find("strand:-") and frame != 0:

self.translated\_id.append(">" + header + "|frame:" + "-" + str(frame))

elif frame == 0:

self.translated\_id.append(">" + header + "|frame:" + str(frame))

self.nuc.append(nucleotide\_string[frame:])

return

def orf\_finder(self):

"""

The functions finds ORF sequences larger than 26 and short than 300 amino acids and writes it in a file.

:return: none

"""

self.orfs = []

self.header\_list = []

count = 0

for x in range(len(self.nuc)):

for y in range(0, len(self.nuc[x]), 3):

if self.nuc[x][y:(y+3)] == "ATG":

start = y

stop = self.nuc[x][start::]

for z in range(0, len(stop), 3):

if stop[z:(z+3)] == "TAA" or stop[z:(z + 3)] == "TAG" or stop[z:(z + 3)] == "TGA":

if len(self.nuc[x][start:start+z+1]) >= (self.min\_protein\_length \* 3)\

and len(self.nuc[x][start:start+z+1]) <= (self.max\_protein\_length \* 3)\

and self.nuc[x][start:start+z]\

not in self.orfs:

self.orfs.append(self.nuc[x][start:start+z])

count += 1

self.header\_list.append(">candidate\_effector:%d|%s|len:%d\n" % (count,

self.translated\_id[x][1::], len(self.nuc[x][start:start+z])))

break

return

def orf\_writer(self):

"""

The function writes the ORFs in a file.

:return: none

"""

orfs\_file = open("ORFS.fasta", "w+")

count = 0

for x in range(len(self.orfs)):

count += 1

orfs\_file.write(self.header\_list[x])

orfs\_file.write(textwrap.fill(str(self.orfs[x].translate()), width=60) + "\n")

orfs\_file.close()

return

def signalp4(self):

"""

The function runs SignalP and finds putative effectors.

:return: none

"""

cline = self.signalp\_path + ' -t euk -f summary -u %s %s > %s' % (self.probability, "ORFS.fasta", "results.txt")

print("Running signalp4.......\n\n\n")

print(cline)

os.system(cline)

file = open("results.txt", "r")

content = file.readlines()

cleavage = []

id\_list = []

if len(open("results.txt").readlines()) == 0:

for file in [snakemake.output.a, snakemake.output.c]:

proteinseq\_file = open(file, "w+")

proteinseq\_file.write("No signalpeptides has been found...")

return

for x in range(len(content)):

if "SP='YES'" in content[x]:

cleavage.append(content[x].split("Cleavage site between pos. ")[1].split(":")[0].replace(" and ", "-"))

id\_list.append(content[x].split("\t")[0].split("Name=")[1])

for file in [snakemake.output.a, snakemake.output.c]:

proteinseq\_file = open(file, "w+")

for x in range(len(self.header\_list)):

for y in range(len(id\_list)):

if ">"+ id\_list[y] == self.header\_list[x].strip():

self.signalpeptideseq\_AA = str(self.orfs[x].translate()[:int(cleavage[y].split("-")[0])])

self.signalpeptideseq = str(self.orfs[x][:int(cleavage[y].split("-")[0])])

if file == snakemake.output.a:

header = self.header\_maker(True, self.header\_list[x])

proteinseq\_file.write("%s\n%s\n" % (header, textwrap.fill((self.signalpeptideseq\_AA.lower()

+ str(self.orfs[x].translate())), width=60)))

if file == snakemake.output.c:

header = self.header\_maker(False, self.header\_list[x])

proteinseq\_file.write("%s\n%s\n" % (header, textwrap.fill(self.signalpeptideseq.lower()

+ str(self.orfs[x]), width=60)))

proteinseq\_file.close()

return

def signalp5(self):

"""

The function runs SignalP and finds putative effectors.

:return: none

"""

os.system("signalp -fasta ORFS.fasta -format short -mature -org 'euk' -prefix 'MetStop'")

os.system("cat MetStop\_summary.signalp5 | egrep 'CS pos' | awk '{if($NF > " + self.probability +

"){print($0)}}' > results.txt")

return

def signalp\_writer(self):

"""

The function writes the results of SignalP in a file.

:return: none

"""

file = open("results.txt", "r")

content = file.readlines()

cleavage = []

print("yolo")

#return

for x in range(len(content)):

cleavage.append(content[x].split(" ")[2][:-1]) #the cleavage of the signalpeptide

content[x] = content[x].split("\t")[0] #id

for file in [snakemake.output.a, snakemake.output.c]:

proteinseq\_file = open(file, "w+")

for x in range(len(self.header\_list)):

for y in range(len(content)):

if ">"+ content[y] == self.header\_list[x].strip():

self.signalpeptideseq\_AA = str(self.orfs[x].translate()[:int(cleavage[y].split("-")[0])])

self.signalpeptideseq = str(self.orfs[x][:int(cleavage[y].split("-")[0])])

if file == snakemake.output.a:

header = self.header\_maker(True, self.header\_list[x])

print(textwrap.fill((self.signalpeptideseq\_AA.lower())))

proteinseq\_file.write("%s\n%s\n" % (header, textwrap.fill((self.signalpeptideseq\_AA.lower()

+ str(self.orfs[x].translate())), width=60)))

if file == snakemake.output.c:

header = self.header\_maker(False, self.header\_list[x])

print(textwrap.fill(self.signalpeptideseq.lower()))

proteinseq\_file.write("%s\n%s\n" % (header, textwrap.fill(self.signalpeptideseq.lower()

+ str(self.orfs[x]), width=60)))

proteinseq\_file.close()

return

def header\_maker(self, boolean, header):

"""

The function adds the correct length of the effector to the current header

:return: returns the new header

"""

header\_list = header.split("|")

length = int(re.search("len:(.\*)", header).group(1))

if boolean:

new\_header = "%s|len:%d" % ("|".join(header\_list[0:4]), ((length / 3) + len(self.signalpeptideseq\_AA)))

elif boolean == False:

new\_header = "%s|len:%d" % ("|".join(header\_list[0:4]), (length + len(self.signalpeptideseq)))

h\_list = new\_header.split("|")

if len(h\_list[0] + "|" + self.signalpeptideseq\_AA + "|" + "|".join(h\_list[1:])) > 60:

sp\_header = h\_list[0] + "|" + self.signalpeptideseq\_AA + "|" + "".join(h\_list[1])

else:

sp\_header = h\_list[0] + "|" + self.signalpeptideseq\_AA + "|" + "|".join(h\_list[1:])

return sp\_header

if \_\_name\_\_ == '\_\_main\_\_':

sample = MetStop(snakemake.input.a)

## Bijlage 9 – augustus.py

import subprocess

import re

import os

import configparser

from Bio import SeqIO

from Bio.Seq import Seq

from Bio.SeqRecord import SeqRecord

from Bio.Alphabet import generic\_dna

from Bio.Alphabet import IUPAC

from Bio.SeqFeature import SeqFeature, FeatureLocation

from BCBio import GFF

import textwrap

class augustusPredictor:

def \_\_init\_\_(self, file):

self.file = file

self.config = snakemake.config

try:

dic = SeqIO.to\_dict(SeqIO.parse(open(self.file), "fasta"))

except:

self.make\_unique\_headers()

self.augustus\_runnen()

self.signalPpath = ""

self.sequentie()

if self.config["Parameters"]["signalp\_version"] == "4":

self.RunSignalP4()

else:

self.RunSignalP5()

#input()

self.ExtractOrfToFasta()

self.write\_out()

self.remove()

def make\_unique\_headers(self):

count = 0

new\_file = open(self.file + "2", "w")

with open(self.file) as f:

for line in f:

if line.startswith(">"):

count = count + 1

new\_header = line.split(" ")

extra\_info = new\_header[-1].split("|")[1]

new\_file.write(new\_header[0] + str(count) + "|" + extra\_info)

else:

new\_file.write(line)

if line is None:

break

new\_file.close()

self.file = self.file + "2"

def augustus\_runnen(self):

"""

Runs Augustus to predict genes

Parameters:

- "self.file: The name of the file with to mimps

- Self.file[:-6]: Name of the file without .fasta

"""

self.directory=self.file.split("/")[1].split(".")[0]

command = ("augustus --species=fusarium --singlestrand=true " +

self.file + " > " + self.directory + "\_augustus.gff")

print("running augustus......")

os.system(command)

def sub\_feature\_extract(self):

"""Loops trought the augustus file and checks if a sentace is a CDS

if that is the case the exonlocation are saved. Returns the CDS sequence

exon locations and the strand (normal or reverse)

Parameters:

- sub\_feature: What augustus found, can be CDS, transcript or stop\_codon

- self.rec: the sequence of the founded gene

- sub\_feature.location.start: the start location of the sequence

- ub\_feature.location.end: The end location of the sequence

"""

self.CDS = ""

self.exonlocations = []

self.feature\_stand = ""

for sub\_feature in self.feature.sub\_features:

if (sub\_feature.type) == "CDS" and sub\_feature.strand == 1:

self.CDS += sub\_feature.extract(self.rec).seq

elif (sub\_feature.type) == "CDS" and sub\_feature.strand == -1:

self.CDS += self.feature.extract(self.rec).seq.reverse\_complement()

if (sub\_feature.type == "CDS"):

self.exonlocations.append(

[int(sub\_feature.location.start), int(self.feature.location.end)])

self.feature\_stand = sub\_feature.strand

def introns\_exons\_extract(self):

"""

Finds the intron locations by looping through exonlocations

intron location is the location between 2 exons. return all\_location and intronlocation

Parameters:

- self.exonlocations: a list with the exonlocations

"""

all\_locations = []

intronlocations = []

for x in range(len(self.exonlocations)):

all\_locations.append(self.exonlocations[x][0])

all\_locations.append(self.exonlocations[x][1])

for y in range(len(self.exonlocations)-1):

intronlocations.append(

[all\_locations[(y\*2)+1], all\_locations[(y+1)\*2]])

self.all\_locations= all\_locations

self.intronlocations = intronlocations[1:-1]

def ORF\_finder(self):

"""

Finds Open reading frame by looping through te exons and introns.

if there are no intron locations the ORF is between the start exon location

and end. Returns the ORF

Parameters:

- self.intronlocations: List of the intronlocations

- self.rec.seq: the sequence of a found gene by augustus

- self.exonlocations: list with the positions of exons

- self.intronlocations: list with the positions of intros

"""

ORF = ""

for x in range(len(self.intronlocations)):

ORF += self.rec.seq[self.exonlocations[x][0]:self.exonlocations[x][1]].upper()

ORF += self.rec.seq[self.intronlocations[x][0]:self.intronlocations[x][1]].lower()

if self.exonlocations != []:

ORF += self.rec.seq[self.exonlocations[len(self.exonlocations)-1]

[0]:self.exonlocations[len(self.exonlocations)-1][1]].upper()

self.ORF = ORF

def header\_info(self):

"""

Splits the header, so the useful information is kept

Parameters:

- self.rec.id: The ID of a predicted gene

- self.feature.location.start: The start of the gene

- self.feature.location.end: The end of the gene

"""

self.contig = self.rec.id.split("|")[0]

self.region\_start = self.rec.id.split(":")[1].split("-")[0]

self.region\_end = self.rec.id.split(":")[1].split("-")[1].split("\_")[0]

if "rev.transc" in self.rec.id:

self.strand = "-"

else:

self.strand = "+"

self.mimp\_IR\_seq = self.rec.id.split("-")[1].split("^")[0]

self.mimp\_IR\_loc = self.rec.id.split("^")[1].split("-")[1]

self.gene\_start\_relative = self.feature.location.start

self.gene\_end\_relative = self.feature.location.end

def real\_orientation\_gen(self):

"""checks if the CDS and ORFS are good orientated. If not so it is changed

Parameters:

- self.feature\_stand: The side the sequence is read, can be + or -

- self.strand: The side the sequence is read, can be + or -

- self.region\_start: the start of the sequence

- self.gene\_start\_relative: the start of the gene

- self.region\_end: end of the sequence

- self.region\_end\_relative: end of the gene"""

self.real\_orientation = ""

self.gene\_start\_absolute = ""

self.gene\_end\_absolute = ""

if self.feature\_stand == 1 and self.strand == "+":

self.real\_orientation = 1 # gene is found on coding strand downstream of MIMP IR

self.gene\_start\_absolute = int(self.region\_start)+int(self.gene\_start\_relative)

self.gene\_end\_absolute = int(self.region\_start)+int(self.gene\_end\_relative)-1

elif self.feature\_stand == 1 and self.strand == "-":

self.real\_orientation = -1 # gene is found on other strand upstream of MIMP IR

self.gene\_start\_absolute = int(self.region\_end)-int(self.gene\_start\_relative)

self.gene\_end\_absolute = int(self.region\_end)-int(self.gene\_end\_relative)+1

elif self.feature\_stand == -1:

self.ORF = self.ORF.reverse\_complement()

self.CDS = self.CDS.reverse\_complement()

if self.strand == "+":

self.real\_orientation = -1

self.gene\_start\_absolute = int(self.region\_start)+int(self.gene\_start\_relative)

self.gene\_end\_absolute = int(self.region\_start)+int(self.gene\_end\_relative)-1

elif self.strand == "-":

self.real\_orientation = 1

self.gene\_start\_absolute = int(self.region\_end)-int(self.gene\_start\_relative)

self.gene\_end\_absolute = int(self.region\_end)-int(self.gene\_end\_relative)+1

def protein\_sequence(self):

"""Checks if CDS is filled (found) if not the case a translated

CDS is returned

Parameters:

- self.CDS: the Coding Sequence of the gene.

"""

if self.CDS == "":

self.proteinseq = "incomplete"

else:

self.proteinseq = self.CDS.translate()[:-1]

def met\_check(self):

"""Check if the translated CDS starts with a methiodine, and if

the protein is longer then 25 AA and shorter then 600 AA. Return

ORFS, CDS, PROT that is in the specific range. The length of protein

are defined in the config file, and can be changed.

Parameters:

- self.proteinseq: CDS sequence translated to a protein sequence

- Self.gene\_start\_relative: the start of the gene

- self.ORF: the Open reading frame of a gene

- self. CDS: the coding sequence of a gene

- self.new\_fastaheader\_id: a new header for the founded CDS

- self.new\_fastaheader\_description: a new description for the header

- min\_len: minimum length of protein, defined in the config file

- max\_len: maximum length of protein, defined in the config file

- d2m: distance that the protein maximum can have from the mimp, defined in the config file

"""

min\_len = self.config["Augustus"]["min\_protein\_length"]

max\_len = self.config["Augustus"]["max\_protein\_length"]

d2m = self.config["Augustus"]["max\_distance\_mimp"]

self.new\_ORF = ""

self.new\_CDS = ""

self.new\_PROT = ""

if str(self.proteinseq[:1]) == 'M' and len(self.proteinseq) > int(min\_len) and len(self.proteinseq) < int(max\_len):

if int(self.gene\_start\_relative) < int(d2m):

self.new\_ORF = (SeqRecord(seq=self.ORF, id=self.new\_fastaheader\_id,

description=self.new\_fastaheader\_description))

self.new\_CDS = (SeqRecord(seq=self.CDS, id=self.new\_fastaheader\_id,

description=self.new\_fastaheader\_description))

self.new\_PROT = (SeqRecord(seq=self.proteinseq, id=self.new\_fastaheader\_id,

description=self.new\_fastaheader\_description))

def feature\_extract(self):

"""Loops through the features of each gen. And connects the different functions.

Returns a list with all the proteins sequence, ORF and CDS

Parameters:

- self.rec.features: Seq info about the founded gene

- self.feature: More info about the founded gene example the location

- self.contig: contig

- self.gene\_start\_absolute: the start of the sequence

- self.gene\_end\_absolute: the start of the sequence

- self.self.gene\_start\_relative: Distance to mimp

- self.real\_orientation: the strand of the sequence

- self.proteinseq: protein sequence

- self.mimp\_IR\_seq: signalpeptide

- self.mimp\_IR\_loc: location

"""

for self.feature in self.rec.features:

a = self.feature.sub\_features

self.sub\_feature\_extract()

self.introns\_exons\_extract()

self.ORF\_finder()

self.header\_info()

self.real\_orientation\_gen()

self.protein\_sequence()

self.new\_fastaheader\_id = self.contig+'|'+str(self.gene\_start\_absolute)+'-'+str(self.gene\_end\_absolute)+'|'+'d2m:'+str(

self.gene\_start\_relative)+'bp|'+str(self.real\_orientation)+'|f?'+'|len:'+str(len(self.proteinseq))

self.new\_fastaheader\_description = str(self.mimp\_IR\_seq)+'^'+self.mimp\_IR\_loc

self.met\_check()

if type(self.new\_ORF) != str:

self.CDS\_list.append(self.new\_CDS)

self.ORF\_list.append(self.new\_ORF)

self.PROT\_list.append(self.new\_PROT)

def sequentie(self):

"""Opens the augustus output file, gives every gen info to the function

feature\_extract by using a loop. From that function is get a compleet list of CDS, PROT and ORF

finaly it writes is to files.

Parameters:

- self.file: the file with the mimps

- self.ORF\_list: A list with al the ORFS and the headers

- self.CDS\_list: A list with al the CDS and the headers

- self.PROT\_list: A list with al the proteins and the headers

"""

self.ORF\_list = []

self.CDS\_list = []

self.PROT\_list = []

in\_seq\_handle = open(self.file)

seq\_dict = SeqIO.to\_dict(SeqIO.parse(in\_seq\_handle, "fasta"))

gff\_file = open(self.directory + "\_augustus.gff")

for self.rec in GFF.parse(gff\_file, base\_dict=seq\_dict):

self.feature\_extract()

#SeqIO.write(self.ORF\_list, snakemake.output.augustus\_ORFS, 'fasta')

SeqIO.write(self.CDS\_list, "CDS", 'fasta') #Augustus CDS file

SeqIO.write(self.PROT\_list, "PROTEIN", 'fasta') #Augustus PROTEIN FILE

self.CDS\_list

def RunSignalP4(self):

SignalPpath = self.config["Parameters"]["signalp\_path"]

probability = self.config["Augustus"]["probability"]

naamlijst = []

self.signalpep\_list = []

self.sig = []

print('// Running SignalP 4.1...')

print("\n....\n........\n........")

cline = SignalPpath +' -t euk -f summary -u %s %s > %s' % (probability, "PROTEIN", "result.txt")

os.system(cline)

print(cline)

file = open("result.txt", "r")

id\_line = file.readlines()

self.cleavage = []

self.id\_line = []

for x in range(len(id\_line)):

try:

self.cleavage.append(id\_line[x].split("between pos. ")[1].split(":")[0].replace(" and ", "-"))

self.id\_line.append(id\_line[x].split("Name=")[1].split("\t")[0])

except:

pass

self.CDS = SeqIO.parse("PROTEIN", 'fasta')

for y in self.CDS:

for x in range(len(self.id\_line)):

if y.id == self.id\_line[x]:

self.seq = str(y.seq)

self.signalpep\_list.append(self.seq[:int(self.cleavage[x].split(

"-")[0])].lower() + self.seq[int(self.cleavage[x].split("-")[0]):])

self.sig.append(self.seq[:int(self.cleavage[x].split("-")[0])])

def RunSignalP5(self):

"""Runs signalP5 and parses it so that the signalpeptide sequence is found

Returns the cleavage positions, signalpeptide sequence, signalpeptide + CDS, and header

Parameters:

- self.signalPpath: the location where signalP is installed

"""

probability = self.config["Augustus"]["probability"]

naamlijst = []

self.signalpep\_list = []

self.sig = []

print ('// Running SignalP 5...')

os.system(self.signalPpath +

"signalp -fasta PROTEIN -format short -mature -org 'euk' -prefix test")

os.system(

"cat test\_summary.signalp5 | egrep 'CS pos' | awk '{if($NF > " + probability +" ){print($0)}}' > result.txt")

file = open("result.txt", "r")

self.id\_line = file.readlines()

self.cleavage = []

for x in range(len(self.id\_line)):

self.cleavage.append(self.id\_line[x].split(" ")[2])

self.id\_line[x] = self.id\_line[x].split("\t")[0]

self.CDS = SeqIO.parse("PROTEIN", 'fasta')

print(self.cleavage, self.id\_line)

for y in self.CDS:

for x in range(len(self.id\_line)):

if y.id == self.id\_line[x]:

self.seq = str(y.seq)

self.signalpep\_list.append(self.seq[:int(self.cleavage[x].split(

"-")[0])].lower() + self.seq[int(self.cleavage[x].split("-")[0]):])

self.sig.append(self.seq[:int(self.cleavage[x].split("-")[0])])

def ExtractOrfToFasta(self):

"""Uses the signalpeptide sequence, CDS and cleavage site to find the putative

effectors. Return effector header and effector sequence

Parameters:

self.id\_line: the id of the lines from the file with the mimps

self.sig: signalpeptide sequence

self.cleavage: cleavage posistion

"""

count = 10000

self.CDS = SeqIO.parse("CDS", 'fasta')

self.effector\_header = []

self.effector\_seq = []

for y in self.CDS:

for x in range(len(self.id\_line)):

if y.id == self.id\_line[x]:

count = count + 1

header = self.id\_line[x].split("|")

if header[3] == "1":

nuc\_seq = str(y.seq)

self.effector\_header.append(str(count)[

1:] + "|" + self.sig[x] + "|" + header[2] + "|" + header[-1]+"|" + header[3])

self.effector\_seq.append(nuc\_seq[:int(self.cleavage[x].split(

"-")[0])].lower() + nuc\_seq[int(self.cleavage[x].split("-")[0]):])

else:

nuc\_seq = str(y.seq.reverse\_complement())

self.effector\_header.append(str(count)[

1:] + "|" + self.sig[x] + "|" + header[2] + "|" + header[-1] + "|" + header[3])

self.effector\_seq.append(nuc\_seq[:int(self.cleavage[x].split(

"-")[0])].lower() + nuc\_seq[int(self.cleavage[x].split("-")[0]):])

def write\_out(self):

"""Writes Signalpeptide sequence to file and writes the putative effectors to file

Parameters:

- self.file: File with the mimps

- self.id\_line: the id of the lines from the file with the mimps

- self.signalpep\_list: list with al the signalpeptides

- self.effector\_seq: list with sequence of al the effectors"""

SP\_file = open(snakemake.output.augustus\_effectors\_prot, "w")

EF\_file = open(snakemake.output.augustus\_effectors\_nuc, "w")

for x in range(len(self.id\_line)):

SP\_file.write(">" + self.id\_line[x] + "\n")

SP\_file.write(re.sub("(.{60})", "\\1\n", str(

self.signalpep\_list[x]), 0, re.DOTALL) + "\n\n")

EF\_file.write(">" + self.effector\_header[x] + "|" + self.id\_line[x].split("|")[0] + "\n")

EF\_file.write(re.sub("(.{60})", "\\1\n", str(

self.effector\_seq[x]), 0, re.DOTALL) + "\n\n")

def remove(self):

"""Removes temporary files"""

os.remove("CDS")

os.system("rm \*.gff")

os.remove("PROTEIN")

if self.config["Parameters"]["signalp\_version"] != "4":

os.remove("result.txt")

os.remove("test\_mature.fasta")

os.system("mv test\_summary.signalp5 " + snakemake.output.augustus\_signalp)

else:

os.system("mv result.txt " + snakemake.output.augustus\_signalp)

if \_\_name\_\_ == '\_\_main\_\_':

sample = augustusPredictor(snakemake.input.founded\_mimps)

## Bijlage 10 – presence\_absence.py

import os, re

import subprocess

import os.path

import configparser

class BlastAndCluster:

def \_\_init\_\_(self):

self.config = snakemake.config

self.file = self.config["Parameters"]["name\_dir"] + "/" + "all\_putative\_effectors\_concatenated\_clustered.fasta"

self.db\_file = self.config["Parameters"]["database\_location"]

self.e\_value\_thresh = str(self.config["Presence\_absence"]["e\_value\_thresh"])

genome\_folder = self.config['Parameters']['input\_folder']

identitie\_thresh = self.config["Presence\_absence"]["ident\_thresh"]

fasta\_files = os.popen("ls " + genome\_folder +" | egrep 'fasta|fa|fna'").read()

genome\_list = fasta\_files.split("\n")[:-1]

self.scores\_for\_genome = {}

for x in range(len(genome\_list)):

self.make\_blast\_database(genome\_list[x], genome\_folder)

self.local\_blast(genome\_list[x])

self.parse\_blastout2(genome\_list[x].split(".")[0], identitie\_thresh)

self.score\_to\_genome(genome\_list)

self.outwrite()

def make\_blast\_database(self, genome, genome\_folder):

"""This function makes a database per genome

Parameters:

- self.db\_file: the path where the database are placed

- genome: 1 genome out of the genome folder """

database\_command = "makeblastdb -in " + genome\_folder + "/" + genome +" -out " + self.db\_file + genome.split(".")[0] +" -parse\_seqids -dbtype nucl"

os.system(database\_command)

print(database\_command)

def local\_blast(self, genome):

print("local\_blast")

self.outfile = self.file.split(".")[0] + "\_VS\_" + genome.split(".")[0] + "\_blast.out"

#blastn -db database/Fol4287sample -query Fol4287sample.fasta -evalue 0.001 -out result.out

os.system("blastn -db " + self.db\_file + genome.split(".")[0] + " -query " + self.file +" -evalue " + self.e\_value\_thresh + " -out " + self.outfile)

def parse\_blastout2(self, genome, identitie\_thresh):

"""Checks in if founded effectors are also found in other genomes. If thats the case

the signalpeptide of the effector is put into a dictionary with the a list of the genome names

Parameters:

- genome: 1 genome out of the genome folder

- self.outfile: The blast file that is created in the local\_blast function

- self.scores\_for\_genome: dictionary with the signalpeptide of each effector, and value of each genome it is present."""

outfile = open(self.outfile, "r")

blastout = outfile.read()

blast\_list = blastout.split("Query=")[1:]

for x in range(len(blast\_list)):

if "\*" not in blast\_list[x]:

scores = blast\_list[x].split("\n")

query = scores[0]

for y in range(len(scores)):

count\_hit = 0

if "Identities" in scores[y]:

identities = int(scores[y].split("%")[0].split("(")[1])

if "Expect = " in scores[y]:

evalue = (scores[y].split("= ")[2])

if "Score =" in blast\_list[x]:

count\_hit = count\_hit + 1

if "Score =" in blast\_list[x] and count\_hit == 2:

print("check")

break

if int(identities) >= int(identitie\_thresh):# and str(evalue) < str(self.e\_value\_thresh):

if query not in self.scores\_for\_genome.keys():

print(query, "toegevoegd")

self.scores\_for\_genome.update({query : [genome]})

else:

update\_value = self.scores\_for\_genome.get(query).append(genome)

def score\_to\_genome(self, genome\_list):

"""this function makes a dictionary, and compares the dic self.scores\_for\_genome

if that dicionary has the genome name in it. The key for the self.score\_dic is a 1

if not the key becomes a 0. Eventually there is a key with a list.

Parameters:

- genome\_list: a list with all the genome names

- self.scores\_for\_genome: dictionary with the signalpeptide of each effector, and value of each genome it is present."""

self.genome\_type = [i.split('.', 1)[0] for i in genome\_list]

self.score\_dic = {}

self.peptide\_string = ""

count = 0

for key, value in self.scores\_for\_genome.items():

count = count + 1

self.peptide\_string = self.peptide\_string + "\t" + str(count) + "." + key

score = []

for x in range(len(self.genome\_type)):

if self.genome\_type[x] in value:

score.append("1")

else:

score.append("0")

self.score\_dic.update({key : score})

def outwrite(self):

"""This funtion makes a tabulair output in the file precence\_absence

Parameters:

- peptide\_string: a string with the signalpeptide and a tab between them.

- self.score\_dic: a dictionary, the key is the signalpeptide and the value is a list with 0 or 1"""

present\_absence = open(snakemake.output.presence\_absence, "w")

present\_absence.write(self.peptide\_string + "\n")

count = 0

total\_list = []

for x in range(len(self.genome\_type)):

total\_list.append([])

for key , value in self.score\_dic.items():

for x in range(len(value)):

total\_list[x].append(value[x])

for x in range(len(total\_list)):

present\_absence.write(self.genome\_type[x] + "\t" + "\t".join(total\_list[x]))

present\_absence.write("\n")

if \_\_name\_\_ == '\_\_main\_\_':

# moet BlastAndCluster(genome\_list[:-1]) worden zodra de rule ervoor klaar is

BlastAndCluster()

## Bijlage 11 – clustering.py

import os

from Bio import SeqIO

import configparser

from Bio import AlignIO

from Bio.Align import AlignInfo

def parse\_header(infile):

compleet\_file = open("testje.fasta", "w")

raw\_file = os.popen("cat " + infile).read().split("\n")

for x in range(len(raw\_file)):

if raw\_file[x].startswith(">") == True:

compleet\_file.write(">" + raw\_file[x].split("|")[1] + "\n")

print(">" + raw\_file[x].split("|")[1] + "\n")

else:

compleet\_file.write(raw\_file[x] + "\n")

compleet\_file.close()

return compleet\_file

def run\_blast(file, dbname):

outfmt\_command = "'6 qseqid sseqid pident qstart qend evalue bitscore qlen'"

# index in outputfile: 0 1 2 3 4 5 6 7

#make blastdb

os\_command\_makedb = 'makeblastdb -in ' + file + ' -dbtype nucl -out ' + dbname + '/' + file.split('.')[0]

print('---------Command to make blastdb:---------\n' + os\_command\_makedb)

os.system(os\_command\_makedb)

#run blastn

os\_command\_blastn = "blastn -outfmt " + outfmt\_command + " -db " + dbname + '/' + file.split('.')[0] + \

' -query ' + file + ' -out blast\_results.out'

print('\n---------Command to blastn:---------\n' + os\_command\_blastn + '\n')

os.system(os\_command\_blastn)

def find\_clusters(PERC\_IDENTITY\_THRESH, LENGTH\_THRESH):

dict\_effector2homologs = {}

with open('blast\_results.out', 'r') as effector\_file:

lines\_in\_file = effector\_file.readlines()

for line in lines\_in\_file:

tabs = line.strip().split('\t')

if float(tabs[2]) > PERC\_IDENTITY\_THRESH and (int(tabs[4]) - (int(tabs[3])-1)) / float(tabs[7]) > LENGTH\_THRESH:

if tabs[0] in dict\_effector2homologs.keys():

dict\_effector2homologs[tabs[0]].add(tabs[1]) # add all BLAST-associated hits to this entry

else:

dict\_effector2homologs[tabs[0]] = set([tabs[1]])

if tabs[1] in dict\_effector2homologs.keys():

dict\_effector2homologs[tabs[1]].add(

tabs[0]) # the other way around; also add the BLAST query as an association to each BLAST hit.

else:

dict\_effector2homologs[tabs[1]] = set([tabs[0]])

return single\_linkage(dict\_effector2homologs)

def single\_linkage(node\_partners):

clusters = []

nodes = list(node\_partners.keys())

while len(nodes) > 0:

nodes = list(node\_partners.keys())

cluster = update\_cluster(node\_partners, nodes[0], set([]))

clusters.append(cluster)

nodes = node\_partners.keys()

return clusters

def update\_cluster(node\_partners, todo, cluster = set([])):

new\_todo = set([])

for t in node\_partners[todo]:

partners = node\_partners[t]

new\_todo = new\_todo.union(set(partners))

cluster.add(t)

del node\_partners[t]

new\_todo = new\_todo.difference(cluster)

if len(new\_todo) > 0:

return update\_cluster(node\_partners, new\_todo, cluster)

else:

return cluster

def find\_longest\_sequence\_in\_cluster(clusters, infile):

#Make a dict of the sequences in the infile. With the headers as keys

with open(infile, 'r') as handle:

record\_dict = SeqIO.to\_dict(SeqIO.parse(handle, "fasta"))

list\_longest\_elements = [] #list is returned with longest sequece of the clusters.

for c in clusters:

print('Nodes in this cluster cluster', len(c))

longest\_element = [0, '', ''] #length, id, sequence

for element in c:

length\_elem = len(record\_dict[element].seq)

if length\_elem > longest\_element[0]:

longest\_element[0] = length\_elem

longest\_element[1] = element

#longest\_element[2] = record\_dict[element].seq

list\_longest\_elements.append(record\_dict[element])

return list\_longest\_elements

def consensus\_sequence(clusters, infile):

with open(infile, 'r') as handle:

record\_dict = SeqIO.to\_dict(SeqIO.parse(handle, "fasta"))

list\_consenses\_elements = [] #list is returned with concensus sequence of the clusters.

for clust in clusters:

print('Nodes in this cluster cluster', len(clust))

if len(clust) > 1:

consensus, id = make\_concensus(clust, record\_dict)

list\_consenses\_elements.append([str(consensus), str(id)])

else:

one\_list = list(clust)

list\_consenses\_elements.append([str(record\_dict[one\_list[0]].seq), str(record\_dict[one\_list[0]].id)])

return list\_consenses\_elements

def make\_concensus(clust, record\_dict):

with open('temp.fasta', 'w') as file:

for elem in clust:

file.write('>' + str(record\_dict[elem].id) + '\n')

file.write(str(record\_dict[elem].seq) + '\n')

id = elem

alignment = AlignIO.read('temp.fasta', 'fasta')

summary\_align = AlignInfo.SummaryInfo(alignment)

concensus = summary\_align.dumb\_consensus()

os.system('rm temp.fasta')

return concensus, id

def write\_long(list\_elements, outputfile):

with open(outputfile, 'w') as file:

for record in list\_elements:

new\_id = record.id.split("|")[1]

record.id = new\_id

record.description = new\_id

SeqIO.write(record, file, 'fasta')

def write\_con(list\_elements, outputfile):

with open(outputfile, 'w') as handle:

for elem in list\_elements:

seq = elem[0]

id = elem[1].split('')[1]

handle.write('>' + id + '\n' + seq + '\n')

def main():

config = configparser.ConfigParser()

config.read('config.ini')

infile = snakemake.input.metstop\_eff\_nuc

PERC\_IDENTITY\_THRESH = int(config['Clustering']['pers\_ident\_thresh'])

LENGTH\_THRESH = float(config['Clustering']['length\_thresh'])

longest\_or\_concensus = config['Clustering']['longest\_or\_concensus']

dbname = config["Parameters"]["database\_location"]

output\_folder = config["Parameters"]["name\_dir"]

outputfile = snakemake.output.eff\_clustered

run\_blast(infile, dbname)

clusters = find\_clusters(PERC\_IDENTITY\_THRESH, LENGTH\_THRESH)

if longest\_or\_concensus == 'longest':

list\_longest\_cluster = find\_longest\_sequence\_in\_cluster(clusters, infile)

write\_long(list\_longest\_cluster, outputfile)

elif longest\_or\_concensus == 'concensus':

list\_consensus\_cluster = consensus\_sequence(clusters, infile)

write\_con(list\_consensus\_cluster, outputfile)

else:

print('Not the right Length\_or\_Concensus variable')

os.system("mv blast\_results.out " + output\_folder + "/clustering\_results.blastout")

main()

## Bijlage 12 – INSTALL.sh

#!/bin/bash

sudo apt-get update

sudo apt-get upgrade

# Install Python3.7

echo "Installing Python3.7..."

sudo add-apt-repository ppa:deadsnakes/ppa

sudo apt-get update

sudo apt install python3.7

sudo apt-get update

# Install pip

echo "Installing pip..."

sudo apt install python3-pip

sudo apt-get update

pip3 install --upgrade pip

# Install biopython3.6

sudo apt-get update

echo "Installing biopython3.6..."

python3 -m pip install biopython

# Install bcbio-gff

sudo apt-get update

echo "Installing bcbio-gff..."

python3 -m pip install bcbio-gff

# Install Snakemake

sudo apt-get update

echo "Installing Snakemake..."

sudo apt-get install snakemake

# Install blast

echo "Installing NCBI-BLAST+..."

sudo apt-get install ncbi-blast+

# Install R

echo "Installing R..."

echo "deb https://cloud.r-project.org/bin/linux/ubuntu disco-cran35/" | sudo tee -a /etc/apt/sources.list

echo "deb https://cloud.r-project.org/bin/linux/ubuntu cosmic-cran35/" | sudo tee -a /etc/apt/sources.list

echo "deb https://cloud.r-project.org/bin/linux/ubuntu bionic-cran35/" | sudo tee -a /etc/apt/sources.list

echo "deb https://cloud.r-project.org/bin/linux/ubuntu xenial-cran35/" | sudo tee -a /etc/apt/sources.list

echo "deb https://cloud.r-project.org/bin/linux/ubuntu trusty-cran35/" | sudo tee -a /etc/apt/sources.list

sudo apt-get update

sudo apt-get install r-base

sudo apt-get install build-essential

sudo apt-get update

sudo apt-get upgrade

## Bijlage 13 – config.ini

[Parameters]

# specify the directory with the genomes

# Folder where all the genomes are saved

input\_folder = genomes/

# Name of the directory where the output data is saved

name\_dir = output\_data

database\_location = databases/

# path to signalp-4.1: [specified\_path\_in\_config\_file]/signalp-4.1/signalp

# path to signalp-5.0: [specified\_path\_in\_config\_file]/signalp-5.0b/bin/signalp

signalp\_path =

# Specify which SignalP version. Type '5' for signalp-5.0 (new) or '4' for signalp-4.1 (older).

signalp\_version = 5

[Mimpfinder]

# Motive where te pipeline looks for. In this cases the MIMP motive.

motive = TT[TA]TTGC..CCCACTG..

# Reverse complentemt of the motive.

motive\_rc = ..CAGTGGG..GCAA[TA]AA

# Must be int. Length of the flanks next to the MIMP motive (or other motive)

length = 2500

[MetStop]

# The minimal probability the signalpeptide sequence has to be.

probability = 0.550

# Minimal length an ORF must have

min\_protein\_length = 26

# Max length an ORF must have

max\_protein\_length = 300

[Augustus]

# The minimal probability the signalpeptide sequence has to be.

probability = 0.550

# Minimal length an ORF must have

min\_protein\_length = 26

# Max length an ORF must have

max\_protein\_length = 300

# Max basepairs of the effector form the MIMP

max\_distance\_mimp = 2000

[Clustering]

# Percentage indentity threshold. If sequences have a higher per\_ident then they are put into a cluster.

# If length\_thresh is also true.

pers\_ident\_thresh = 60

# Blast output: '(qend - qstart) / qlen'. If the threshold is smaler then '(qend - qstart) / qlen' the sequences

# are put into a cluster. If pers\_ident\_thresh is also true.

length\_thresh = 0.3

# The pipeline chooses 1 sequece from the cluster. Longeste takes the longeste sequence of the cluster

# and concensus makes a concensus sequence.

longest\_or\_concensus = longest

[Presence\_absence]

e\_value\_thresh = 0.001

ident\_thresh = 30

## Bijlage 14 – Snakefile

import configparser

import os

config = configparser.ConfigParser()

config.read("config.ini")

directory = config['Parameters']['name\_dir']

input\_folder = config['Parameters']['input\_folder']

genome\_files = os.popen("ls " + input\_folder +" | egrep 'fasta|fa|fna|ffn|faa|frn'").read()

genomes = genome\_files.split("\n")[:-1]

subdir = [i.split('.', 1)[0] for i in genomes]

print(subdir)

rule all:

input:

expand("{directory}/{subdir}/{subdir}\_mimp\_flank.fasta", subdir=subdir, directory=directory,),

expand("{directory}/{subdir}/{subdir}\_MetStop\_putative\_effectors\_nuc.fasta", subdir=subdir, directory=directory),

expand("{directory}/{subdir}/{subdir}\_Augustus\_putative\_effectors\_aminoacid.fasta", subdir=subdir, directory=directory),

directory + "/combined\_effectors.fasta",

expand("{directory}/all\_putative\_effectors\_concatenated\_clustered.fasta",directory=directory),

expand("{directory}/presence\_absence.txt", directory=directory),

expand("{directory}/blastn\_presence\_absence\_reordered.txt", directory=directory)

rule mimp\_fasta:

input:

a="config.ini",

b=input\_folder + "{subdir}.fasta"

output:

a="{directory}/{subdir}/{subdir}\_mimp\_flank.fasta",

b="{directory}/{subdir}/{subdir}\_tir\_output.fasta",

script:

"MimpFinder.py"

rule mimp\_fa:

input:

a="config.ini",

b=input\_folder + "{subdir}.fa"

output:

a="{directory}/{subdir}/{subdir}\_mimp\_flank.fasta",

b="{directory}/{subdir}/{subdir}\_tir\_output.fasta",

script:

"MimpFinder.py"

rule mimp\_fna:

input:

a="config.ini",

b=input\_folder + "{subdir}.fna"

output:

a="{directory}/{subdir}/{subdir}\_mimp\_flank.fasta",

b="{directory}/{subdir}/{subdir}\_tir\_output.fasta",

script:

"MimpFinder.py"

rule MetStop:

input:

a="{directory}/{subdir}/{subdir}\_mimp\_flank.fasta"

output:

a="{directory}/{subdir}/{subdir}\_MetStop\_putative\_effectors\_AA.fasta",

b="{directory}/{subdir}/{subdir}\_MetStop\_summary.signalp5",

c="{directory}/{subdir}/{subdir}\_MetStop\_putative\_effectors\_nuc.fasta"

script:

"MetStop.py"

rule augustus:

input:

founded\_mimps="{directory}/{subdir}/{subdir}\_mimp\_flank.fasta"

output:

augustus\_signalp="{directory}/{subdir}/{subdir}\_Augustus\_summary.signalp5",

augustus\_effectors\_prot="{directory}/{subdir}/{subdir}\_Augustus\_putative\_effectors\_aminoacid.fasta",

augustus\_effectors\_nuc="{directory}/{subdir}/{subdir}\_Augustus\_putative\_effectors\_nucleotide.fasta"

script:

"augustus.py"

rule combine:

input:

file1 = expand("{directory}/{subdir}/{subdir}\_Augustus\_putative\_effectors\_nucleotide.fasta", directory=directory, subdir=subdir),

file2 = expand("{directory}/{subdir}/{subdir}\_MetStop\_putative\_effectors\_nuc.fasta", directory=directory, subdir=subdir)

output:

"{directory}/combined\_effectors.fasta"

shell:

"""

cat {input.file1} {input.file2} > {output}

"""

rule clustering:

input:

metstop\_eff\_nuc="{directory}/combined\_effectors.fasta"

output:

eff\_clustered ="{directory}/all\_putative\_effectors\_concatenated\_clustered.fasta"

script:

"clustering\_0108.py"

rule presence\_absence\_effectors:

input:

clustered\_effectors ="{directory}/all\_putative\_effectors\_concatenated\_clustered.fasta"

output:

presence\_absence ="{directory}/presence\_absence.txt"

script:

"presence\_absence.py"

rule visualization:

input:

presence\_absence = "{directory}/presence\_absence.txt"

output:

reordered = "{directory}/blastn\_presence\_absence\_reordered.txt"

shell:

"Rscript 04.cluster\_and\_plot\_heatmap3.R heatmap.3.R {input.presence\_absence} {directory}/ 1 average 1 average"

## Bijlage 15 – README.txt

This program is made by: Date: Januari 10 2020

Aron van Beelen

Timo de Graaf

Aldo Vree

Lieke Vree

Mentor: Like Fokkens (University of Amsterdam)

This is the README of the pipeline that finds effectors in genomes of fungi. The pipeline only works in a Unix based System e.g. MacOS or Ubuntu (preferably Ubuntu).

------------------------

# INSTALLATION GUIDE

------------------------

1. Manually install SignalP-4.1, SignalP-5.0 and Augustus3.3

a. Open the 'Signalp\_Tools' directory in the navigator.

b. Open the terminal in the 'SignalP\_Tools' directory and type:

'tar -xvzf signalp-5.0b.Linux.tar.gz signalp-5.0b'

'tar -xvzf signalp-4.1g.Linux.tar.gz signalp-4.1'

'tar -xzf augustus.current.tar.gz'

or (for MacOS)

'tar -xvzf signalp-5.0b.Darwin.tar.gz signalp-5.0b'

'tar -xvzf signalp-4.1g.Darwin.tar.gz signalp-4.1'

'tar -xzf augustus.current.tar.gz'

This results in two directories: 'signalp-4.1', 'signalp-5.0b' and augustus-3.3.3.

c. Move the three directories to a desired path.

For example: '/home/user/tools'.

Later on, this is the path that you have to refer to in the config file.

d. An extra step for signalp-4.1 has to be taken. Move to the signalp-4.1 directory (in the specified path) and open the 'signalp' file.

In the 'GENERAL SETTINGS', change the full path to the signalp-4.1 directory (in the specified path) on your system:

For example: '/home/user/tools/signalp-4.1'

e. An extra step for augustus has to be taken. Move to the augustus-3.3.3 directory (in the specified path) and open the terminal. Type 'make' and hit Enter.

Wait. Type sudo make install and hit Enter.

'

2. Run INSTALL.sh

INSTALL.sh is a bash file which installs the requirements to use the pipeline. It downloads and/or installs:

BioPython3.6, bcbio-gff, Augustus, Snakemake, R and blastn.

a. Go back to the pipeline directory and open the terminal in that directory.

b. Type 'bash INSTALL.sh' in the terminal and hit Enter.

\* Type password and type 'y' if necessary.

3. Manually install R packages

a. Type 'R' in the terminal and hit Enter. This opens the R terminal.

b. Type and hit Enter:

\*install.packages("dendextend")

Question: select CRAN mirror --> choose a desired location. (We picked UK London).

\*install.packages("gplots")

\*install.packages("extrafont")

\*install.packages("ade4")

\*if (!requireNamespace("BiocManager", quietly = TRUE))

install.packages("BiocManager")

BiocManager::install(version = "3.10")

\*BiocManager::install("ctc")

------------------------

# PIPELINE USAGE GUIDE

------------------------

The pipeline runs with Snakemake; see installation guide above.

First, change the parameters in the config file. Read the commentary in the config file carefully.

1. Open the terminal on your computer (Unix based system).

2. Move to the directory where the pipeline is stored. Directory = ????

3. Change, if necessary, the parameters in the config file.

4. Type 'snakemake' in the terminal and hit enter.

# The config.ini file

In the config file are all the parameters of the pipeline. If you want to change a parameter, change it in the config

file and run the pipeline again. Al the possible options for each parameter are given in the comments.