Uniwersytet Przyrodniczy we Wrocławiu
Wydział Biologii i Hodowli Zwierząt
Kierunek: Bioinformatyka
Studia stacjonarne pierwszego stopnia

Paweł Tomkowski
121562

Stworzenie narzędzia do automatycznego procesowania składania sekwencji mtDNA pochodzącego z sekwencjonowania NGS, na przykładzie sekwencji genomowych ślimaków z rodzaju Trochulus

Creation of a tool for automatic assembling of mtDNA sequences from NGS using genome sequences of snails from genus Trochulus

Praca wykonana pod kierunkiem
Dr. Tomasza Strzały
Katedra Genetyki

Wrocław, 2022

Oświadczenie opiekuna pracy

Oświadczam, że niniejsza praca została przygotowana pod moim kierownictwem i stwierdzam, że spełnia ona warunki do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data ……. Podpis opiekuna pracy …….

Oświadczenie autora pracy

Oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami ani też nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Data ……. Podpis autora pracy …….

Streszczenie

Stworzenie narzędzia do automatycznego procesowania składania sekwencji mtDNA pochodzącego z sekwencjonowania NGS, na przykładzie sekwencji genomowych ślimaków z rodzaju Trochulus

W ramach projektu zostało stworzone narzędzie mające na celu ułatwić szerszej grupie naukowców dostęp do odtwarzania sekwencji mtDNA ze względnie szeroko dostępnych danych. Zbierając istniejące już programy i narzędzia powstał skrypt który względnie efektywnie automatycznie przetwarza dane z NGS typu Illumina na sekwencję mitochondrialną.

Słowa kluczowe: NGS, mtDNA, Trochulus, Bioinformatyka, bash

Abstract

Creation of a tool for automatic assembling of mtDNA sequences from NGS using genome sequences of snails from genus Trochulus

As part of the project a tool has been created to allow more scientists access to recreation of mtDNA from readily accessible data. Gathering already existing programs and tools a script was created which quite effectively atomically process Ilumina type NGS data to mitochondrial sequence.

Keywords: NGS, mtDNA, Trochulus, Bioinformatics, bash

Spis treści:

1. Wstęp.

Human Genome Project oraz technologie, które się wraz z nim pojawiły pokazały, że wielkoskalowe sekwencjonowanie nawet dużych genomów z dobrą dokładnością jest możliwe. W kolejnych latach różne firmy i organizacje zajęły się rozwijaniem tego konceptu, tworząc coraz to szybsze, efektywniejsze oraz co najważniejsze tańsze urządzenia i techniki do sekwencjonowania. Tak narodziło się NGS – Sekwencjonowanie Nowej Generacji. NGS to zbiorczy termin określający zbiór urządzeń i technologii pozwalających na sekwencjonowanie całego dużego genomu (zwierzęcego/roślinnego) w ciągu kilku dni, a w przypadku najbardziej zaawansowanych urządzeń i technologii nawet kilku dziennie[1]. Stało się to możliwe głównie dzięki znacznemu zautomatyzowaniu i miniaturyzacji całego procesu. Najsłynniejszy przedstawiciel tego typu technologii oraz wzorzec, do którego porównywano wszystkie inne, to urządzenia firmy Illumina. Głównym ograniczeniem tej technologii jest to, iż odczyt następuje w małych elementach (do 200 par zasad), które trzeba potem złożyć w kompletny genom, co z racji dużej liczby wszelkiego rodzaju powtórzeń w genomach eukariotycznych utrudnia pracę z tymi danymi. Pewnym rozwiązaniem było sekwencjonowanie tego samego materiału kilkukrotnie, aby osiągnąć nałożenie pomiędzy poszczególnymi krótkimi odczytami. Na przestrzeni ostatnich lat pojawiły się nowe technologie pozwalające na znacznie dłuższe odczyty, głównie z firmy Oxford Nanopore. Odczyty ich urządzeń mają średnio kilka tysięcy par zasad długości, co rozwiązuje wiele problemów związanych z krótkimi odczytami, a najbardziej zaawansowane wersje pozwalają nawet na odczyty długości ponad 2 mln pz[2].

Z czasem uświadomiono sobie także, że skoro sekwencjonujemy cały genom z komórki eukariotycznej, to wśród uzyskanych danych będzie także genom mitochondrialny oraz chloroplastowy (w przypadku roślin). Wykorzystując pewne algorytmy można próbować odtworzyć te genomy na podstawie całościowego sekwencjonowania DNA komórki. Dane z tych genomów są cenne, gdyż mają inną budowę i w związku z tym ewoluują inaczej niż genom jądrowy (szybciej) oraz względnie niezależnie od niego. Pozwala to na wykorzystanie ich do analizy systematycznej pomiędzy różnymi organizmami. Powstały więc wyspecjalizowane narzędzia odtwarzające genom mitochondrialny z sekwencji całościowej, jednakże zwykle wymagają one odpowiednich kroków, zanim będzie można ich użyć na danych z maszyn sekwencjonujących.

Istnieją protokoły[3], które opisują krok po kroku sposób, w jaki można wykorzystać istniejące narzędzia, głównie MitoFinder oraz NOVOplasty do tego aby wydobyć informację o genomie mitochondrialnym z odczytu całego genomu. Były one inspiracją do opracowania niniejszego narzędzia , które miało zautomatyzować ten proces, aby zwiększyć dostępność tego typu analiz dla większej liczby naukowców. Opracowany skrypt będzie miał zastosowanie dla osób, które mają mniejsze doświadczenie lub umiejętności w obcowaniu z programowaniem. Mam nadzieję, że powiększy to naszą wiedzę na temat genomów mitochondrialnych, ich ewolucji oraz systematyki organizmów .

2. Materiały i metody

Dane użyte w projekcie stanowiły odczyty genomu ślimaka z rodzaju Trochulus wykonane przez firmę CeGaT GmbH. Biblioteka została stworzona za pomocą zestawu TruSeq DNA PCR-Free (Ilumina). Odczyt został wykonany za pomocą sekwencera NovaSeq 6000, przy użyciu kuwety przepływowej typu 2 x 150 pz. Wartość Q30 $\geq$ 88,25%. Sekwencje adapterowe zostały odcięte za pomocą programu Skewer (wersja 0.2.2). Nie były przeprowadzone żadne przycięcia związane z jakością odczytów. Dało to dwa pliki sparowane sumarycznie dające 798 324 tysięcy odczytów, 118 944 milionów par zasad.

W toku analizy zostały użyte następujące programy oraz skrypty:

- system typu Linux - pomimo wygody i znajomości systemu Windows w większości zastosowań specjalistycznych system ten nie sprawdza się, używany jest system typu Linux,

który jest bardziej funkcjonalny i przez to większość specjalistycznych programów jest stworzona dla niego;

- bash – powłoka systemów UNIX, pośrednik pomiędzy systemem Linux a zainstalowanymi programami, także domyślny język do kontrolowania wszystkiego w systemie, dlatego bardzo wygodny do pisania skryptów mających kontrolować inne programy, większość programu jest w nim napisana;
  - sudo apt-get install – funkcja używana do instalacji programów głównie z internetowych repozytoriów, zawsze z opcją „sudo" dającą uprawnienia administracyjne, aby móc dokonywać zmian w systemie,
  - mkdir – tworzy nowy katalog o zadanej nazwie, w zadanej lokalizacji,
  - cd – zmiana obecnego katalogu na inny, co zmienia miejsce odniesienia pewnych komend,
  - git clone – tworzy kopię zadanego repozytorium z platformy github i pobiera ją do zadanego folderu,
  - sed - funkcja służąca do zastępowania pewnych znaków innymi znakami w plikach tekstowych, mało wydajna w dużej skali,
  - function – tworzy funkcje, które mogą być później wywoływane,
  - echo – funkcja, która pokazuje to co ma przekazane, używana do wyświetlania informacji albo przekazywania tekstu dla innych funkcji,
  - grep – komenda służąca do wyszukiwania i wyświetlania linii z pasującym wzorcem w tekście, w programie używany z opcją -c do podania tylko liczby wierszy z pasującym wzorcem.
- AWK – interpretowalny język stworzony i wykorzystywany do różnego typu operacji na tekście, głównie wyszukiwania i przetwarzania konkretnych wzorców;
- cURL - funkcja do wysyłania i odbierania plików z internetu używając do tego składni URL;
- perl - jeden z języków programowania, dużo programów bioinformatycznych ma część skryptów, albo jest w całości napisana w tym języku;
- docker - program ułatwiający włączanie różnych programów poprzez stworzenie i włączanie tak zwanych kontenerów - samodzielnych miniaturowych systemów operacyjnych wraz ze wszystkimi potrzebnymi funkcjami i programami potrzebnymi do aktywacji oryginalnego programu; pozwala to na używanie różnych programów bez względu na istniejący system operacyjny albo zainstalowane funkcje w systemie;
- fastqc[4] - narzędzie do kontroli jakości danych z sekwencjonowania; zawarty w skrypcie jako przydatne narzędzie do sprawdzenia jakości posiadanych danych;
- MitoFinder[5] – program do ekstrakcji danych mitochondrialnych z sekwencjonowania nowej generacji; został użyty, gdyż był w oryginalnym protokole;
- NOVOplasty[6] – program do odtwarzania genomów organellowych z całych genomów; został użyty, gdyż był w oryginalnym protokole;
- fastp[7] – program do oczyszczania i prostego przetwarzania plików z sekwencjonowania; został użyty, gdyż był potrzebny szybki program do oczyszczania danych z sekwencjonowania;
- seqkit[8] – pakiet do analizy i przetwarzania plików z sekwencjonowania; został użyty, do szybkiej analizy rozmiaru plików;
- MITObim[9] – program do odtwarzania genomu mitochondrialnego z sekwencji całego genomu, w programie używany w dockerze; przez A.Hiley użyty z racji posiadania funkcji downsample.py, która pozwala na próbkowanie danych; w skrypcie program został dodany jako kolejny do ekstrakcji mtDNA;
- BBmap[10] – program do przyrównywania globalnego sekwencji DNA i RNA; w SUNmito.sh wykorzystywany jest tylko skrypt, który pozwala na rozplecenie pliku, w którym zostały razem zapisane odczyty sparowane, jest to wykorzystywane, aby analiza była prowadzona na dwóch plikach sparowanych;

Wyjaśnienie działania programu:

Program jest uruchamiany za pomocą wiersza poleceń w powłoce bash systemu Linux poprzez wpisanie komendy ./SUNmito.sh z odpowiednimi zmiennymi. Link do repozytorium, w którym znajduje się program: https://github.com/Jenlotis/praca_lic. Ogólna komenda do uruchomienia programu: $ ./SUNmito.sh -A -i ścieżka/do/folderu -m ścieżka/do/referencji/mitofindera.gb -n ścieżka/do/referencji/novoplasty.fasta -b ścieżka/do/referencji/mitobim.fasta -O numer_organizmu_według_mitofindera -r ilość_RAM-u

Do pełnego uruchomienia jest potrzebne:

- podanie ścieżki, która ma być wykorzystana(-M MitoFinder, -N NOVOplasty, -B MITObim -A wszystkie 3) powoduje to ustalenie wartości zmiennej "alfa" jako M/N/B/A zależnie od wybranej ścieżki i to wartość tej zmiennej określa uruchamianą ścieżkę zapisaną w formie funkcji (w ten sposób flaga programu może być w każdym miejscu)

```
305         -B)
306                 #echo "Both programs are running"
307                 alfa=B
308                 shift
309                 ;;
310         -M)
311                 #echo "only MITOfinder is running"
312                 alfa=M
313                 shift
314                 ;;
315
316         -N)
317                 #echo "only NOVOPlasty is running"
318                 alfa=N
319                 shift
320                 ;;
```

;

- podanie pełnej ścieżki do folderu (-i) z sekwencjami w formacie fastq spakowane (rozszerzenie .fastq.gz) program automatycznie wyszukuje wszystkie pliki, które spełniają te warunki i dodaje je do listy, która będzie używana do uruchomienia kolejnych podprogramów;
- podanie pełnej ścieżki do sekwencji referencyjnych (-m dla MitoFindera (format genbank), -n dla NOVOplasty (format fasta), -b dla MITObim (format fasta)), których użyją programy do tego aby spróbować odtworzyć genom mitochondrialny, program za pomocą funkcji "grep" z flagą "-c" sprawdza ilość linii z charakterystycznymi znakami dla wymaganego formatu ("LOCUS" dla formatu genbank oraz ">" dla formatu fasta), jeżeli program znajdzie jedną taką linię plik jest uznawany za zgodny z formatem (nawet jeżeli plik przedostanie się przez ten filtr i tak zostanie odrzucony przy wywołaniu programu składającego, jednak nawet taki filtr pozwoli na szybsze pozbycie się prostych błędów);
- w przypadku używania programu MitoFinder potrzebne jest także podanie typu organizmu (-O) (a w ten sposób także kodu genetycznego), od którego są dane, pełna lista jest w pomocy dla MitoFinder-a, w przypadku wzorcowych danych było to 5 dla bezkręgowców;
- ilość RAM-u do użycia przez NOVOplasty (-r), nie jest bezpośrednio potrzebne ale autorzy sugerują użycie jej, gdy nie posiadamy serwerowej ilości RAM-u (rozmiar pakietu danych +~10%);

pozostałe flagi to:

- pomoc (-h) informacje na temat programu i działania flag, jest małą funkcją, która jest uruchamiana przy pojawieniu się flagi w wywołaniu programu, która pokazuje tekst w terminalu. Po pokazaniu tekstu program jest zamykany;
- parowalność (-p) czy sekwencje są sparowane czy nie (**T**(true)/**F**(false)), określa to jakie subprogramy będą wywoływane oraz sposób przetwarzania plików z folderu wejściowego:
  - w przypadku podania opcji T zmienna, która zawiera nazwy plików z folderu powstaje poprzez wyciągniecie za pomocą komendy "ls" nazw wszystkich plików ze wskazanego katalogu. Dane te są przekazane do AWK, który wyszukuje wszystkie pliki zawierające w swojej nazwie ".1.fastq.gz"(w takiej formie był zapisany plik, na którym SUNmito.sh było testowane oraz prawdopodobnie większość sprawowanych odczytów), ta lista jest przekazana do kolejnego AWK, który odrzuca wcześniej wymienioną końcówkę (założenie programowe: plik nazywa się NAZWY.1.fastq.gz jeżeli owa NAZWY zawiera w sobie kropki program nie będzie w stanie poprawnie odczytać tego pliku), aby pozostała sama NAZWY, która jest przekazywana jako główna zmienna do programów;

```
330   if [ $PAROWALNOSC = T ]
331   then
332           # reads names of every set of input data
333           NAZWY=$(ls $wejscie |awk '$0 ~ ".1.fastq.gz" {print $0}' | awk -F "." '{print $1}')
334           echo $NAZWY
```

  - w przypadku podania opcji F AWK poszukuje wszystkich plików zawierających ".fastq.gz", z racji tego, iż jest to szerokie poszukiwanie - znajdzie także pliki wykorzystywane przy opcji T, potrzebne jest ich odrzucenie. Dokonywane jest to przez przekazanie danych do kolejnego AWK, który sprawdza czy plik ma pomiędzy NAZWY a ".fastq.gz" wartość 1 lub 2, jeżeli tak ten plik jest odrzucany, kolejne AWK zostawia samą NAZWY, która jest przekazana jako główna zmienna do programów;

```
350   elif [ $PAROWALNOSC = F ]
351   then
352           # reads names of every set of input data
353           NAZWY=$(ls $wejscie |awk '$0 ~ ".fastq.gz" {print $0}' |awk -F "." '$2 !~ "1" && $2 !~ "2" '| awk -F "." '{print $1}')
354           echo $NAZWY
```

- zainstalowanie (-Z) wszystkich wymaganych programów oraz pobranie wszystkich wymaganych repozytoriów z platformy github, jest to stworzone jako funkcja, która jest uruchamiana przy pojawieniu się flagi podczas wywoływania programu. Używana komenda "sudo apt-get install --assume-yes" - jako super użytkownik/administrator zainstaluj/aktualizuj pakiet ze znanego repozytorium, załóż że użytkownik się zgadza(komenda po znalezieniu pakietu pyta się użytkownika czy na pewno chce go zainstalować, aby upłynnić użytkowanie używane jest to stwierdzenie):
  - fastqc,
  - curl,
  - za pomocą biblioteki curl dodawany jest dostęp do repozytorium brew,
  - z repozytorium brew jest pobierany seqkit (nie jest wymagane sudo z racji miejsca instalacji seqkit),
  - fastp,
  - libidn11 - biblioteki wykorzystywane przez MitoFinder, na nowszych wersjach linuxa nie są domyślnie zainstalowane,
  - interpreter języka perl potrzebny dla novoplasty z racji tego iż program jest napisany w tym języku,
  - Python - wiele skryptów w wykorzystywanych programach jest napisanych w Pythonie dlatego jest aktualizowany do najnowszych wersji,
    - python2 - wymagane przez wiele skryptów, nowsze instalacje linuxa nie posiadają,
    - pip - menedżer instalacji pakietów do Pythona,
    - python3 - najnowsza wersja Pythona jaka istnieje,

        - pip specyficzny dla pythona3,
    - cmake - otwarty system pomagający w kompilacji programów,
    - git - otwarty system kontroli wersji oraz dużo narzędzi w tym pomocnych, w programie używany jako "clone", aby pobrać kopię repozytorium z wymaganymi programami z platformy github,
    - mawk - jeden z interpreterów dla AWK,
    - BBmap,
    - automake autoconf - dwa pakiety odpowiedzialne za ułatwienie i uproszczenie kompilacji programów,
    - mkdir ./github - tworzy folder o nazwie github w obecnym folderze roboczym,
    - cd ./github - zmienia folder roboczy na nowo stworzony folder,
    - git clone - pobiera repozytorium z github-a do którego został podany link,
        - MITObim,
        - NOVOplasty,
        - MitoFinder,
            - za pomocą komendy "cd" zostaje zmieniony folder roboczy na ten, w którym znajduje się MitoFinder, "./instal.sh" instaluje ów program, a "p=$(pwd)" "echo -e "\n#Path to MitoFinder image \nexport PATH=\$PATH:$p" >> ~/.bashrc" "source ~/.bashrc" dodaje program do zmiennej PATH, która zawiera w sobie ścieżki, w pierwszej kolejności sprawdzane przy wywoływaniu komendy w bash,

```
42          git clone https://github.com/RemiAllio/MitoFinder.git
43                  cd MitoFinder
44                  ./install.sh
45                  p=$(pwd)
46                  echo -e "\n#Path to MitoFinder \nexport PATH=\$PATH:$p" >> ~/.bashrc
47                  source ~/.bashrc
```

    - po wykonaniu wszystkiego zamyka program;
- ilość rdzeni(-t), którą może wykorzystać fastp oraz seqkit, w przypadku nieużycia flagi programy użyją domyślnych wartości.

Ścieżki to zbiory różnych funkcji/subprogramów zebrane razem, aby pozwolić na dość modularne wykorzystywanie funkcji dla każdego rozpoznanego pliku w folderze:

1. MitoFinder
    - pair end
        - clean_pair
            - za pomocą instrukcji warunkowej jest sprawdzana obecność oczyszczonych plików dla tych danych,
                - jeżeli nie, to są tworzone oczyszczone za pomocą fastp,
                - jeżeli tak, to krok zostanie pominięty;
            - struktura "$wejscie$i" pozwala na automatyczne odczytywanie pasujących plików z folderu; "-i" oraz "-I" są wejściowymi plikami, a "-o" i "-O" są już oczyszczonymi danymi, które są zapisywane do osobnego folderu dla większej przejrzystości;
            - w skrypcie są używane domyślne wartości związane z oczyszczaniem plików;
                - zależnie od danych, te wartości będą inne i trudno określić jakie będą wymagane do konkretnego pliku;
        - downsam_p2p
            - funkcja całkowicie stworzona pod MitoFinder, w przyszłości będzie możliwe uogólnienie jej funkcjonalności;

- zadaniem tej funkcji jest stworzenie próbki danych za pomocą skryptu downsample.py z pakietu MITObim, a potem z racji tego, że skrypt zapisuje próbkę do jednego pliku rozplecenie danych za pomocą skryptu reformat.sh z pakietu BBmap;
- subprogram za pomocą instrukcji warunkowej sprawdza czy istnieją już próbki plików wejściowych,
    - jeżeli tak, to sprawdza za pomocą funkcji ls(oraz kilku formatowań za pomocą grep oraz AWK) do jakich wartości procentowych zostały próbkowane, po czym informuje użytkownika o istniejących plikach i pyta się czy użytkownik chce wykorzystać jeden z istniejących czy stworzyć nowy plik.
        - W przypadku chęci użycia istniejącego pliku pokazywana jest lista wszystkich istniejących wersji, po czym użytkownik jest pytany o podanie jednej z wymienionych wartości. Po jej wybraniu program zapisuje je jako zmienna możliwa do użycia przez kolejne programy;
        - W przypadku chęci stworzenia nowego pliku program przechodzi do próbkowania;
    - jeżeli nie, to przechodzi bezpośrednio do próbkowania;
- MitoFinder nie działa optymalnie, gdy ma przeprowadzić analizę na więcej niż na 7 000 000 odczytach sparowanych. Dlatego koniecznym krokiem jest stworzenie próbki z oczyszczonych danych za pomocą odpowiedniej funkcji, jednak zanim to nastąpi potrzebne jest sprawdzenie jaki rozmiar próbki jest potrzebny:
    - dzieje się to za pomocą seqkit, który oblicza podstawowe statystyki z jednego z oczyszczonych już plików,
    - całe statystyki są przekazane do AWK, który wyodrębnia z tabeli dane na temat długości,
    - z racji tego, iż format używany przez seqkit ma przecinki jako separator tysięcy musi zostać użyty sed, który zastąpi wszystkie przecinki niczym, co pozwoli na przeprowadzenie operacji arytmetycznych na danych,
    - używając AWK, posiadana wartość zostaje porównana z maksymalną i zamieniona na wartość procentową. Program informuje użytkownika o tym jaki procent wyszedł z takiej analizy, po czym podaje rekomendowaną wartość(odrzucenie miejsc dziesiętnych z racji tego, iż przyjmowana wartość przez kolejny program może być tylko liczbą całkowitą a zaokrąglanie szczególnie przy dużych danych może mocno odbiegać od maksymalnej wartości) i pyta się jakiej wartości użytkownik chce użyć. Po podaniu wartości program poinformuje użytkownika o wartości jaką podał i przejdzie do następnego etapu.
- za pomocą funkcji downsample.py z pakietu MITObim jest pobierana próbka danych o rozmiarze podanym przez użytkownika, z oczyszczonych plików. Z racji tego, że wyjściem funkcji jest tylko jeden plik, jest używana opcja "--interleave", aby poprawnie go zapisać, jest on od razu przekazywany funkcji gzip, która go spakuje dla oszczędności miejsca,

        - za pomocą skryptu reformat.sh z pakietu BBmap stworzony w poprzednim kroku plik jest rozkładany do dwóch osobnych plików sparowanych, flaga "int=t" upewnia się, iż plik będzie interpretowany jako dwa pliki wymieszane ze sobą,
    - mitfi_pair
        - po odpowiednim przygotowaniu danych, można uruchomić samego MitoFinder-a, przez wpisanie kolejnych flag:
            - nazwa projektu "-j" (dla łatwiejszej identyfikacji) to nazwa pliku wraz z procentem próby, "-1" oraz "-2" to rozplecione pliki z poprzedniego subprogramu, "-r" to plik referencyjny(jak wcześniej wspomniano w formacie genbank), oraz "-o" typ organizmu podany przy wywoływaniu programu, plikiem wyjściowym najważniejszym dla tego skryptu jest "[Seq_ID]_mtDNA_contig.fasta" który zawiera odtworzony genom mitochondrialny w formacie fasta, pozostałe pliki wyjściowe to: informacje na temat budowy odtworzonej sekwencji oraz jakie informacje zostały użyte do odtworzenia sekwencji;
- single end
    - clean_sing
        - za pomocą instrukcji warunkowej jest sprawdzana obecność oczyszczonego pliku dla tych danych,
            - jeżeli nie to jest tworzony oczyszczony za pomocą fastp,
            - jeżeli tak to krok zostanie pominięty;
        - struktura "$wejscie$i" pozwala na automatyczne odczytywanie pasujących plików z folderu,
            - "-i" jest wejściowym plikiem, a "-o" jest już oczyszczonymi danymi które są zapisywane do osobnego folderu dla większej przejrzystości;
        - w skrypcie są używane domyślne wartości związane z oczyszczaniem plików;
            - zależnie od danych, te wartości będą inne i trudno określić jakie będą wymagane do konkretnego pliku;
    - downsam_s2s
        - funkcja całkowicie stworzona pod MitoFinder, w przyszłości będzie możliwe uogólnienie jej funkcjonalności;
        - zadaniem tej funkcji jest stworzenie próbki danych za pomocą skryptu downsample.py z pakietu MITObim;
        - subprogram za pomocą instrukcji warunkowej sprawdza czy istnieje już próbka pliku wejściowego,
            - jeżeli tak, to sprawdza za pomocą funkcji ls(oraz kilku formatowań za pomocą grep oraz AWK) do jakich wartości procentowych został próbkowany, po czym informuje użytkownika o istniejących plikach i pyta się czy użytkownik chce wykorzystać jeden z istniejących czy stworzyć nowy plik.
                - W przypadku chęci użycia istniejącego pliku pokazywana jest lista wszystkich istniejących wersji, po czym użytkownik jest pytany o podanie jednej z wymienionych wartości. Po jej wybraniu program zapisuje je jako zmienna możliwa do użycia przez kolejne programy;

            - W przypadku chęci stworzenia nowego pliku program przechodzi do próbkowania;
        - jeżeli nie, to przechodzi bezpośrednio do próbkowania;
    - MitoFinder nie działa optymalnie gdy ma przeprowadzić analizę na więcej niż na 7 000 000 odczytów sparowanych(14 000 000 niesparowanych) dlatego koniecznym krokiem jest stworzenie próbki z oczyszczonych danych za pomocą odpowiedniej funkcji, jednak zanim to nastąpi potrzebne jest sprawdzenie jaki rozmiar próbki jest potrzebny,
        - dzieje się to za pomocą seqkit, który oblicza podstawowe statystyki oczyszczonego pliku,
        - całe statystyki są przekazane do AWK, który wyodrębnia z tabeli dane na temat długości,
        - z racji tego, iż format używany przez seqkit ma przecinki jako separator tysięcy musi zostać użyty sed, który zastąpi wszystkie przecinki niczym, co pozwoli na przeprowadzenie operacji arytmetycznych na danych,
        - używając AWK posiadana wartość zostaje porównana z maksymalną i zamieniona na wartość procentową, program informuje użytkownika o tym jaki procent wyszedł z takiej analizy, po czym podaje rekomendowaną wartość(odrzucenie miejsc dziesiętnych z racji tego, iż przyjmowana wartość przez kolejny program może być tylko liczbą całkowitą a zaokrąglanie szczególnie przy dużych danych może mocno odbiegać od maksymalnej wartości) i pyta się jakiej wartości użytkownik chce użyć, po podaniu wartości program poinformuje użytkownika o wartości jaką podał i przejdzie do następnego etapu,
    - za pomocą funkcji downsample.py z pakietu MITObim jest pobierana próbka danych o rozmiarze podanym przez użytkownika, z oczyszczonego pliku, próbka jest od razu kompresowana dla oszczędności miejsca za pomocą funkcji gzip;
- mitfi_sing
    - po odpowiednim przygotowaniu danych możliwe jest teraz uruchomienie samego MitoFinder-a,
    - nazwa projektu "-j" (dla łatwiejszej identyfikacji) to nazwa pliku wraz z procentem próby, "-s" plik z poprzedniego subprogramu, "-r" to plik referencyjny(jak wcześniej wspomniano w formacie genbank), oraz "-o" typ organizmu podany przy wywoływaniu programu, plikiem wyjściowym najważniejszym dla tego skryptu jest "NAZWY.PROCENT_mtDNA_contig.fasta", który zawiera odtworzony genom mitochondrialny w formacie fasta, pozostałe pliki wyjściowe to: informacje na temat budowy odtworzonej sekwencji oraz jakie informacje zostały użyte do odtworzenia sekwencji.

2. NOVOplasty
    - paired end
        - novpla
            - z racji wygody i zabezpieczenia przed używaniem złego pliku wzorcowego do modyfikacji, funkcja ma w sobie cały plik konfiguracyjny, który jest tworzony za każdym razem przed włączeniem programu i jest on używany do jego wywołania,
            - w pliku modyfikowane są linie:

        - “Project name” - nazwa projektu, używane do nazywania plików,
        - “Forward reads” i “Reverse reads” - ścieżka do plików wejściowych,
        - “Seed Input” - ścieżka do referencji,
        - “Output path” - ścieżka gdzie mają zostać zapisane pliki wyjściowe,
        - pozostałe pozostają bez zmian,
      - program jest uruchamiany jest przez wywołanie skryptu NOVOPlastyVERSION.pl z opcją -c wskazującą na stworzony plik konfiguracyjny za pomocą interpretera perl.
  - single end
    - novoplasty nie wspiera opcji używania plików single end (opcja jednak istnieje w pliku konfiguracyjnym);
3. MITObim
  - paired end
    - interleave
      - z racji tego że mitobim ma problem z wieloma plikami, pliki sparowane muszą zostać połączone do jednego pliku, w sposób który zachowa tą informację,
      - używając funkcji reformat.sh z pakietu bbmap następuje zapisanie danych z obydwu plików sparowanych do jednego,
    - mitobim_pair
      - tworzony jest aktywny kontener z całym MITObim-em w trybie pozwalającym na przekazywanie do działającego już kontenera komend (możliwość komunikacji z kontenerem z poziomu skryptu) w przypadku pierwszego uruchomienia komendy, kontener jest pobierany z odpowiedniego repozytorium; tworzone są także synchronizacje pozwalające na przekazanie kontenerowi danych oraz późniejsze ich wyjęcie,
      - do zmiennej zostaje zapisana nazwa kontenera (potrzebna informacja, aby móc przekazać odpowiedniemu kontenerowi komendy),
      - za pomocą komendy docker exec jest aktywowany MITObim w kontenerze,
      - dane wyjściowe w kontenerze zostają skopiowane do odpowiedniego zsynchronizowanego folderu,
      - niepotrzebny już kontener zostaje zatrzymany i usunięty,
  - single end
    - mitobim_sing
      - tworzony jest aktywny kontener z całym MITObim-em w trybie pozwalającym na przekazywanie do działającego już kontenera komend (możliwość komunikacji z kontenerem z poziomu skryptu) w przypadku pierwszego uruchomienia komendy, kontener jest pobierany z odpowiedniego repozytorium; tworzone są także synchronizacje pozwalające na przekazanie kontenerowi danych oraz późniejsze ich wyjęcie,
      - do zmiennej zostaje zapisana nazwa kontenera (potrzebna informacja, aby móc przekazać odpowiedniemu kontenerowi komendy),
      - za pomocą komendy docker exec jest aktywowany MITObim w kontenerze,
      - dane wyjściowe w kontenerze zostają skopiowane do odpowiedniego zsynchronizowanego folderu,

- niepotrzebny już kontener zostaje zatrzymany i usunięty,

W przypadku wybrania opcji -A wszystkie wymienione programy są uruchamiane po kolei, zależnie od flagi -p T/F w wersji single albo pair end.

3. Wyniki

Ostatecznym wyjściem dla każdego z tych programów jest odtworzony genom mitochondrialny możliwy do użycia w dalszych analizach. Zależnie od uruchomionego programu poza odtworzonym genome w zdefiniowanych lokalizacjach znajdą się także inne pliki zawierające więcej informacji na temat analizy:

- MitoFinder
  - NAZWY.PROCENT_final_genes_NT.fasta - zawiera ostateczne sekwencje nukleotydowe genów znalezionych w odtworzonych genomach mitochondrialnych
  - NAZWY.PROCENT_final_genes_AA.fasta - zawiera ostateczne sekwencje aminokwasowe genów znalezionych w odtworzonych genomach mitochondrialnych
  - NAZWY.PROCENT_mtDNA_contig.fasta - odtworzony genom mitochondrialny w formacie fasta
  - NAZWY.PROCENT_mtDNA_contig.gff - odtworzony genom mitochondrialny z przewidzianym umiejscowieniem genów w formacie GFF3
  - NAZWY.PROCENT_mtDNA_contig.tbl - odtworzony genom mitochondrialny z przewidzianym umiejscowieniem genów w formacie zdatnym do opublikowania w Genbank-u
  - NAZWY.PROCENT_mtDNA_contig.gb - odtworzony genom mitochondrialny z przewidzianym umiejscowieniem genów w formacie wizualizacyjnym Genbank-u
  - NAZWY.PROCENT_mtDNA_contig_genes_NT.fasta - zawiera sekwencje nukleotydowe przewidzianych genów
  - NAZWY.PROCENT_mtDNA_contig_genes_AA.fasta - zawiera sekwencje aminokwasowe przewidzianych genów
  - NAZWY.PROCENT_mtDNA_contig.png - schemat przedstawiający umiejscowienie przewidzianych genów na odtworzonym genomie mitochondrialnym
  - NAZWY.PROCENT_mtDNA_contig.infos - zawiera dane na temat odtworzonego genomu: długość, nazwę oraz zawartość GC
- NOVOplasty
  - Contigs_NAZWY.txt - wszystkie możliwe odtworzone genomy: całościowe bądź fragmentaryczne
  - Circularized_assembly_NAZWY.fasta - próba odtworzenia kolistego genomu z jednego z odtworzonych: fragmentarycznych bądź całościowych
  - Merged_contigs_NAZWY.txt - jeżeli żaden z odtworzonych fragmentów nie jest odpowiedni do odtworzenia kolistego genomu, NOVOPlasty postara się połączyć je ze sobą, aby odtworzyć kolisty genom
- MITObim
  - w folderze ./NAZWY/output są dwa foldery nazwane iteration* w tych folderach znajduje się
    - folder NAZWY-NAZWY_assembly
      - NAZWY_d_results - zawiera odtworzony genom mitochondrialny w różnych formatach,
      - NAZWY_d_info - różne informacje techniczne na temat tego jak program funkcjonował,
      - NAZWY_d_tmp - log-i oraz tymczasowe pośrednie pliki z odtwarzanym genomem,

        - NAZWY_d_chkpt - wszystkie pliki potrzebne do wznowienia analizy w przypadku awarii programu albo chęci jej wydłużenia po zakończeniu jeżeli wyniki nie są zadowalające.
    - backbone_it*_initial_NAZWY.fna - zawiera wybraną sekwencję referencyjną,
    - baitfile.fasta - sekwencja referencyjna w wygodniejszym dla programu ułożeniu,
    - hashstat.bin - prawdopodobnie statystyki dotyczące tablicy haszującej w formie binarnej,
    - manifest.conf - konfiguracyjny plik wejściowy,
    - NAZWY-readpool-it*.fastq - sekwencje z pliku wejściowego uznane za nadające się próby odtworzenia genomu mitochondrialnego,
    - NAZWY-NAZWY-it*_noIUPAC.fasta - sekwencja konsensusowa bez konwencji IUPAC;

4. Podsumowanie

Celem projektu było opracowanie programu, który ułatwi i usprawni pracę wielu badaczy i na podstawie przeprowadzonych testów można stwierdzić, iż stworzony program działa poprawnie. Planowane jest rozwinięcie programu w ramach pracy magisterskiej: wprowadzenie obsługi dłuższych odczytów, większej ilości programów oraz porównywania jakości odtworzonych sekwencji zależnie od programu i typu danych wejściowych. Obecność tego narzędzia ułatwi pracę oraz zdobywanie danych wielu naukowcom oraz pozwoli na poszerzenie naszej wiedzy o mitochondrialnym DNA wykorzystując już istniejące dane z NGS.

5. Literatura

[1] Berglund, E.C., Kiialainen, A. & Syvänen, AC. Next-generation sequencing technologies and applications for human genetic history and forensics. Investig Genet 2, 23 (2011). https://doi.org/10.1186/2041-2223-2-23

[2] Payne, A., Holmes, N., Rakyan, V.K., & Loose, M.W. BulkVis: a graphical viewer for Oxford nanopore bulk FAST5 files. Bioinformatics, 35, 2193 - 2198 (2019).

[3] https://www.protocols.io/view/mitogenome-assembly-from-ngs-genome-skimming-data-5qpvo5j3xl4o/v1

[4] Andrews, S. (2010). FastQC: A Quality Control Tool for High Throughput Sequence Data [Online]. Available online at: http://www.bioinformatics.babraham.ac.uk/projects/fastqc/

[5] Allio, R, Schomaker-Bastos, A, Romiguier, J, Prosdocimi, F, Nabholz, B, Delsuc, F. MitoFinder: Efficient automated large-scale extraction of mitogenomic data in target enrichment phylogenomics. Mol Ecol Resour. 20, 892– 905 (2020). https://doi.org/10.1111/1755-0998.13160

[6] Dierckxsens N., Mardulyn P. and Smits G. NOVOPlasty: De novo assembly of organelle genomes from whole genome data. Nucleic Acids Research, (2016). doi: 10.1093/nar/gkw955

[7] Shifu Chen, Yanqing Zhou, Yaru Chen, Jia Gu; fastp: an ultra-fast all-in-one FASTQ preprocessor. Bioinformatics, 34, 17, i884–i890 (2018). https://doi.org/10.1093/bioinformatics/bty560

[8] Shen W, Le S, Li Y, Hu F (2016) SeqKit: A Cross-Platform and Ultrafast Toolkit for FASTA/Q File Manipulation. PLoS ONE 11(10): e0163962. https://doi.org/10.1371/journal.pone.0163962

[9] Hahn C., Bachmann L., Chevreux B. Reconstructing mitochondrial genomes directly from genomic next-generation sequencing reads—a baiting and iterative mapping approach. Nucleic Acids Research, 41, 13, e129 (2013). https://doi.org/10.1093/nar/gkt371

[10] https://jgi.doe.gov/data-and-tools/software-tools/bbtools/bb-tools-user-guide/bbmap-guide/

SUNmito.sh

```
#!/bin/bash

usage() {
	cat <<EOF
Options
	-h this help
	-Z instalation of all neded programs, nessesery github repositories are unpacked to new folder called 'gihub'
	-i input path to folder with every file
	-p do read are paired, deafult='T'
	-A run all available subprograms
	-M run MitoFinder path
	-m full path to reference file for MitoFinder(genebank format)
	-O type of organism genetic code chceck 'MitoFinder -h' for all options
	-N run NOVOplasty path
	-n full path to reference file for NOVOplasy(fasta format)
	-B run MITObim path
	-b full path to reference file for MITObim

EOF

}

function programy() {

	# installs every necessary program(or update them if allready installed)

	sudo apt-get install --assume-yes fastqc
	sudo apt-get install --assume-yes curl
	/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
	echo 'eval "$(/home/linuxbrew/.linuxbrew/bin/brew shellenv)"' >> /home/$USER/.profile
  eval "$(/home/linuxbrew/.linuxbrew/bin/brew shellenv)"
	brew install seqkit
	sudo apt-get install --assume-yes fastp
	sudo apt-get install --assume-yes libidn11
	sudo apt-get install --assume-yes docker.io
	sudo apt-get install --assume-yes perl
	sudo apt-get install --assume-yes python-pip python3 python3-pip
	sudo apt-get install --assume-yes cmake
	sudo apt-get install --assume-yes git
	sudo apt-get install --assume-yes mawk
	sudo apt-get install --assume-yes bbmap
	sudo apt-get install automake autoconf

	mkdir ./github
	cd ./github

	git clone https://github.com/chrishah/MITObim.git
	git clone https://github.com/Edith1715/NOVOplasty.git
```

```
        git clone https://github.com/RemiAllio/MitoFinder.git
                cd MitoFinder
                ./install.sh
                p=$(pwd)
                echo -e "\n#Path to MitoFinder \nexport PATH=\$PATH:$p" >> ~/.bashrc
                source ~/.bashrc

}

function clean_sing() {

  # chcecking existance of cleaned files for single end data and cleaning them if they don't exist

  if [[ $( ls ./$i/cleaned/ | grep -c $i ) != 1 ]];
        then
                # -i input, -o output, -w amount of used threads,  -V log info every milion bases
                fastp -i $wejscie$i.fastq.gz -o ./cleaned/$i.Out.fastq.gz $THREADf -V
        fi
}

function clean_pair() {

  # chcecking existance of cleaned files for paired end data and cleaning them if they don't exist

  if [[ $( ls ./$i/cleaned/ | grep -c $i.Out..f ) != 2 ]];
        then
                # -i input1, -I input2, -o output1, -O output2, -w amount of used threads,  -V log info
every milion bases
                fastp -i $wejscie$i.1.fastq.gz -I $wejscie$i.2.fastq.gz  -o ./$i/cleaned/$i.Out1.fastq.gz
-O ./$i/cleaned/$i.Out2.fastq.gz $THREADf -V
        fi
}

function downsam_s2s() {

  # downsampling data if: there is no donsampled files/user want to create new one

  if [[ $( ls ./downsampling/ | grep -c $i.downsam_ ) != 0 ]];
  then
    procenty=$( ls ./downsampling/ | grep $i.downsam_ | awk -F "." '{print $2}' | awk -F "_" '{print
$2}' )
    echo "there allready are downsampled files from $i to $procenty %"
    echo "do you want to use existing one or create new one(Existing/New)"
    read CCC
    if [[ $CCC = E ]];
    then
      echo "what percent ($procenty)"
      read XXX
      echo $XXX
    elif [[ $CCC = N ]];
    then
      DOWN=Start
```

```
   fi
  else
   DOWN=Start
  fi

  if [[ $DOWN = Start ]];
  then
   # chcecking size of the file for downsapling
            XXX=$( seqkit stats ./cleaned/$i.Out.fasta $THREADs | awk -v dolari="$i" '$1~"./cleaned/"dolari".Out1.fasta" {print $4}' | sed 's/,//g' | awk '{print 7000000/$1*100}' )
    echo $XXX "this is percent of reads that is closest to the highest for mitofinder, we suggest using " $(printf '%.0f' $XXX) " it is however possible to use lower value(int only)"
   echo "To what percent you want to dowsample(recomended $(printf '%.0f' $XXX)): "
   read XXX
   #XXX=9 #${XXX%.*}
   echo "We will downsaple to $XXX % of the original"

   # downsampling and packing
    # -s percent of the original , --interleave creates one file with paried ends, -r input files, \ gzip > packing and saving to file
    python2 ./github/MITObim/misc_scripts/downsample.py -s $XXX -r ./cleaned/$i.Out.fasta | gzip > ./downsampling/$i.downsam_$XXX.fastq.gz
  fi
}

function downsam_p2p() {

  # input: 2 cleaned files form ilumina; output: 2 downsapled files;
  # if there are already downsampled files function will ask user if they want to use existing ones or create new ones,

  if [[ $( ls ./$i/downsampling/ | grep -c $i.down_pair ) -ge 2 ]];
  then
     procenty=$( ls ./$i/downsampling/ | grep $i.downsam_ | awk -F "." '{print $2}' | awk -F "_" '{print $2}' )
   echo "there allready are downsampled files from $i to $procenty %"
   echo "do you want to use existing one or create new one(Existing/New)"
   read CCC
   if [[ $CCC = E ]];
   then
    echo "what percent ($procenty)"
    read XXX
    echo $XXX
   elif [[ $CCC = N ]];
   then
    DOWN=Start
   fi
  else
   DOWN=Start
  fi

  if [[ $DOWN = Start ]];
```

```
  then
    # chcecking size of the file for downsapling
    date +%T
          XXX=$( seqkit stats ./$i/cleaned/$i.Out1.fastq.gz $THREADs | awk -v dolari="$i" '$1~"./"dolari"/cleaned/"dolari".Out1.fastq.gz" {print $4}' | sed 's/,//g' | awk '{print 7000000/$1*100}' )
     echo $XXX "this is percent of reads that is closest to the highest for mitofinder, we suggest using " $(printf '%.0f' $XXX) " it is however possible to use lower value(int only)"
    echo "To what percent you want to dowsample(recomended $(printf '%.0f' $XXX)): "
    read XXX
    #XXX=9 #${XXX%.*}
    echo "We will downsaple to $XXX % of the original"

    # downsampling i packing
     # -s percent of the original , --interleave creates one file with paried ends, -r input files, \ gzip > packing and saving to file
            python2 ./github/MITObim/misc_scripts/downsample.py -s $XXX --interleave -r ./$i/cleaned/$i.Out1.fastq.gz -r ./$i/cleaned/$i.Out2.fastq.gz | gzip > ./$i/downsampling/$i.downsam_$XXX.fastq.gz

    # deinterlaving file
    # in= input file(interlaved), out1= and out2= out files(seperated paired ends)
                        reformat.sh int=t in=./$i/downsampling/$i.downsam_$XXX.fastq.gz out1=./$i/downsampling/$i.down_pair$XXX.1.fastq.gz out2=./$i/downsampling/$i.down_pair$XXX.2.fastq.gz overwrite=true
  fi
}

function interlave() {

          if [[ $( ls ./$i/cleaned/ | grep -c $i.Out_inter ) != 1 ]];
          then
                    #interlave
                    reformat.sh in1=./$i/cleaned/$i.Out1.fastq.gz in2=./$i/cleaned/$i.Out2.fastq.gz out=./$i/cleaned/$i.Out_inter.fastq.gz overwrite=true
          fi
}


function mitfi_sing() {

  # MITOfinder looking for mitRNA
   # -j process name(internal ID), -s input file single end, -r reference sequence, -o which genetic code to use(5-Invertebrate(bezkregowce))
        python2 mitofinder -j $i.$XXX -s ./downsampling/$i.downsam_$XXX.fastq.gz -r $REFERENCE_M -o $ORGANISM --override
}

function mitfi_pair() {

  # MITOfinder looking for mitRNA
```

```
  # -j process name(internal ID), -1 i -2 input files pair end(-s allows for single end), -r reference sequence, -o which genetic code to use(5-Invertebrate(bezkregowce))
                python2 ./github/MitoFinder/mitofinder -j $i.$XXX -1 ./downsampling/$i.down_pair$XXX.1.fastq.gz -2 ./downsampling/$i.down_pair$XXX.2.fastq.gz -r $REFERENCE_M -o $ORGANISM --override
}

function novpla() {

        # NOVOplasty looking for mitRNA
        # creating config file

        echo "Project:
-----------------------
Project name          = $i
Type                  = mito
Genome Range          = 12000-22000
K-mer                 = 33
Max memory            = 14
Extended log          = 0
Save assembled reads  = no
Seed Input            = $REFERENCE_N
Extend seed directly  = no
Reference sequence    =
Variance detection    =
Chloroplast sequence  =

Dataset 1:
-----------------------
Read Length           = 151
Insert size           = 300
Platform              = illumina
Single/Paired         = PE
Combined reads        =
Forward reads         = $wejscie$i.1.fastq.gz
Reverse reads         = $wejscie$i.2.fastq.gz
Store Hash            =

Heteroplasmy:
-----------------------
MAF                   =
HP exclude list       =
PCR-free              =

Optional:
-----------------------
Insert size auto      = yes
Use Quality Scores    = no
Output path           = $p/$i/


Project:
```

Project name = Choose a name for your project, it will be used for the output files.
Type = (chloro/mito/mito_plant) \"chloro\" for chloroplast assembly, \"mito\" for mitochondrial assembly and
\"mito_plant\" for mitochondrial assembly in plants.
Genome Range = (minimum genome size-maximum genome size) The expected genome size range of the genome.
Default value for mito: 12000-20000 / Default value for chloro: 120000-200000
If the expected size is know, you can lower the range, this can be useful when there is a repetitive
region, what could lead to a premature circularization of the genome.
K-mer = (integer) This is the length of the overlap between matching reads (Default: 33).
If reads are shorter then 90 bp or you have low coverage data, this value should be decreased down to 23.
For reads longer then 101 bp, this value can be increased, but this is not necessary.
Max memory = You can choose a max memory usage, suitable to automatically subsample the data or when you have limited
memory capacity. If you have sufficient memory, leave it blank, else write your available memory in GB
(if you have for example a 8 GB RAM laptop, put down 7 or 7.5 (don\'t add the unit in the config file))
Extended log = Prints out a very extensive log, could be useful to send me when there is a problem (0/1).
Save assembled reads = All the reads used for the assembly will be stored in seperate files (yes/no)
Seed Input = The path to the file that contains the seed sequence.
Extend seed directly = This gives the option to extend the seed directly, in stead of finding matching reads. Only use this when your seed
originates from the same sample and there are no possible mismatches (yes/no)
Reference (optional) = If a reference is available, you can give here the path to the fasta file.
The assembly will still be de novo, but references of the same genus can be used as a guide to resolve
duplicated regions in the plant mitochondria or the inverted repeat in the chloroplast.
References from different genus haven\'t beeen tested yet.
Variance detection = If you select yes, you should also have a reference sequence (previous line). It will create a vcf file
with all the variances compared to the give reference (yes/no)
Chloroplast sequence = The path to the file that contains the chloroplast sequence (Only for mito_plant mode).
You have to assemble the chloroplast before you assemble the mitochondria of plants!

Dataset 1:

Read Length = The read length of your reads.
Insert size = Total insert size of your paired end reads, it doesn\'t have to be accurate but should be close enough.
Platform = illumina/ion - The performance on Ion Torrent data is significantly lower
Single/Paired = For the moment only paired end reads are supported.
Combined reads = The path to the file that contains the combined reads (forward and reverse in 1 file)
Forward reads = The path to the file that contains the forward reads (not necessary when there is a merged file)

Reverse reads = The path to the file that contains the reverse reads (not necessary when there is a merged file)
Store Hash = If you want several runs on one dataset, you can store the hash locally to speed up the process (put \"yes\" to store the hashes locally)
To run local saved files, goto te wiki section of the github page

Heteroplasmy:

MAF = (0.007-0.49) Minor Allele Frequency: If you want to detect heteroplasmy, first assemble the genome without this option. Then give the resulting
sequence as a reference and as a seed input. And give the minimum minor allele frequency for this option
(0.01 will detect heteroplasmy of >1%)
HP exclude list = Option not yet available
PCR-free = (yes/no) If you have a PCR-free library write yes

Optional:

Insert size auto = (yes/no) This will finetune your insert size automatically (Default: yes)
Use Quality Scores = It will take in account the quality scores, only use this when reads have low quality, like with the
300 bp reads of Illumina (yes/no)
Output path = You can change the directory where all the output files wil be stored.)" > ./$i/$i_Nconfig.txt

```
	# all things are in config file
	perl ./github/NOVOplasty/NOVOPlasty4.3.1.pl -c ./$i/$i\_Nconfig.txt


}

function mitobim_sing() {

	# starts a docker(if docker doesn't exist ona a computer crates it) then run MITObim, after mitobim end
	sudo docker run -d -it -v $p/$i/cleaned/:/home/data/input/ -v $p/$i/output/:/home/data/output/ -v $p/reference/:/home/data/reference/ chrishah/mitobim /bin/bash

	kontener=$( sudo docker ps | awk '$0 ~ "chrishah" {print $1}' )

	sudo docker exec $kontener /home/src/scripts/MITObim.pl -sample $i -ref $i -readpool /home/data/input/$i.Out.fastq.gz --quick /home/data/reference/$REFERENCE_B -end 10 --clean

	sudo docker exec $kontener cp -r ./iteration* ./data/output/

	sudo docker stop $kontener
	sudo docker rm $kontener
}

function mitobim_pair() {
```

```
	# starts a docker(if docker doesn't exist ona a computer crates it) then run MITObim, after mitobim end
	sudo docker run -d -it -v $p/$i/cleaned/:/home/data/input/ -v $p/$i/output/:/home/data/output/ -v $p/reference/:/home/data/reference/ chrishah/mitobim /bin/bash

	kontener=$( sudo docker ps | awk '$0 ~ "chrishah" {print $1}' )

	sudo docker exec $kontener /home/src/scripts/MITObim.pl -sample $i -ref $i -readpool /home/data/input/$i.Out_inter.fastq.gz --quick /home/data/reference/$REFERENCE_B -end 10 --clean --redirect_tmp /home/data/output/

	sudo docker exec $kontener cp -r ./iteration* ./data/output/

	sudo docker stop $kontener
	sudo docker rm $kontener
}

alfa=alfa
PAROWALNOSC=T
p=$(pwd)

# jezeli nie ma argumentu(jezeli lista argumentow ma dlugosc 0) wyswietl manual
while test $# -gt 0;
do

	case $1 in
	-h | --help)
		usage
		exit 0
		;;

	-Z)
		echo "installing programs"
		programy
		exit 0
		;;

	-i)
		echo "input path to folder"
		wejscie=("$2")
		shift
		shift
		;;

	-p)
		echo "paired ends"
		PAROWALNOSC="$2"
		shift
		shift
		;;

	-m)
```

```
		echo "reference for MITOfinder"
		REFERENCE_M="$2"
		dana=$( grep -c LOCUS $REFERENCE_M )
		#echo $dana
		if [ $dana != 1 ]
		then
			echo "wrong file format for reference file for MITOfinder should be GenBank format"
			exit 3
		else
			echo
		fi
		shift
		shift
		;;

	-n)
		echo "reference for NOVOPlasty"
		REFERENCE_N="$2"
		dana=$( grep -c ">" $REFERENCE_N )
		#echo $dana
		if [ $dana != 1 ]
		then
			echo "wrong file format for reference file for NOVOPlasty should be fasta format"
			exit 3
		else
			echo
		fi
		shift
		shift
		;;

	-b)
		echo "reference for MITObim"
		REFERENCE_B=$2
		dana=$( grep -c ">" $REFERENCE_B )
		#echo $dana
		if [ $dana != 1 ]
		then
			echo "wrong file format for reference file for MITObim should be fasta format"
			exit 3
		else
			echo
			REFERENCE_B=$( echo $REFERENCE_B | awk -F "/" '{print $3}')
		fi
		shift
		shift
			;;

	-t)
```

```
		echo "thread used $2"
		THREADf="-w $2"
		THREADs="-j $2"
		shift
		shift
		;;

	-O)
		echo "organism"
		ORGANISM="$2"
		shift
		shift
		;;

	-A)
		echo "All programs are running"
		alfa=A
		# mitfi
		# novpla
		shift
		;;

	-B)
		echo "MITObim is running"
		alfa=B
		shift
		;;

	-M)
		echo "MITOfinder is running"
		alfa=M
		# mitfi
		shift
		;;

	-N)
		echo "NOVOPlasty is running"
		alfa=N
		# novpla
		shift
		;;

	esac
done

if [ $PAROWALNOSC = T ]
then
	# reads names of every set of input data
	NAZWY=$(ls $wejscie |awk '$0 ~ ".1.fastq.gz" {print $0}' | awk -F "." '{print $1}')
	echo $NAZWY

	for i in $NAZWY;
```

```
	do

		mkdir $i
		mkdir ./$i/downsampling
		mkdir ./$i/cleaned
		mkdir ./$i/output

		if [ $alfa == A ];
		then
			clean_pair
			downsam_p2p
			mitfi_pair
			novpla
			interlave
			mitobim_pair
		elif [ $alfa == M ];
		then
			clean_pair
			downsam_p2p
			mitfi_pair
		elif [ $alfa == N ];
		then
			novpla
		elif [ $alfa == B ];
		then
			interlave
			mitobim_pair
		else
			echo "program not chosen"
		fi
	done

elif [ $PAROWALNOSC = F ]
then
	# reads names of every set of input data
	NAZWY=$(ls |awk '$0 ~ ".fastq.gz" {print $0}' |awk -F "." '$2 !~ "1" && $2 !~ "2" '| awk
-F "." '{print $1}')
	echo $NAZWY

	for i in $NAZWY;
	do

		mkdir $i
		mkdir ./$i/downsampling
		mkdir ./$i/cleaned
		mkdir ./$i/output

		if [ $alfa == A ];
		then
			clean_sing
			downsam_s2s
			mitfi_sing
```

```
                mitobim_sing
        elif [ $alfa == M ];
        then
                clean_sing
                downsam_s2s
                mitfi_sing
        elif [ $alfa == B ];
        then
                mitobim_sing
        else
                echo "program not chosen"
        fi
    done

fi

exit 1
```