Genome alignment using AnchorWave

Baoxing Song

2021-08-24

Contents

1 Perform genome alignment between zebrafish and goldfish

Here, we are trying to explain and demonstrate how to use the AnchorWave to align the goldfish genome assembly against the zebrafish reference genome.
Firstly, we prepare the input files, the zebrafish reference genomes in fasta format, the zebrafish reference genome annotation in gff(3) format, and the goldfish query genome file in fasta format.

1.1 Download genome sequences and reference GFF file

```
wget ftp://ftp.ensembl.org/pub/release-102/fasta/danio_rerio/dna/Danio_rerio.GRCz11.dna.primary_assembly.fa.gz
wget ftp://ftp.ensembl.org/pub/release-102/gff3/danio_rerio/Danio_rerio.GRCz11.102.chr.gff3.gz
wget https://research.nhgri.nih.gov/goldfish/download/carAur01.sm.fa
gunzip *gz
```

1.2 Check the collinearity and whole-genome duplication

Since AnchorWave uses collinearity blocks and whole genome duplications to guide the genome alignment.
Before running the alignment, we use minimap2 to map reference full-length CDS to query genome, visualize the mapping and get ideas about the collinearity and whole-genome duplication.

1.2.1 Overview the genome assemblies

Firstly, we indexed the genome files using samtools.

```
samtools faidx carAur01.sm.fa
samtools faidx Danio_rerio.GRCz11.dna.primary_assembly.fa
```

Then we check the name and sequence length for each sequence record. By looking into the generated file "carAur01.sm.fa.fai" and "Branchiostoma_lanceolatum.BraLan2.dna.toplevel.fa.fai" manually, we roughly know they are chromosome level genome assemblies(not contig level). To check collinearity and whole-genome duplication, we only need to focus on chromosomes. They are `"1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13", "14", "15", "16", "17", "18", "19", "20", "21", "22", "23", "24", "25", "MT"` for "Branchiostoma_lanceolatum.BraLan2.dna.toplevel.fa"
and `"LG1","LG2","LG3", "LG4", "LG5", "LG6", "LG7", "LG8", "LG9", "LG10", "LG11", "LG12", "LG13", "LG14", "LG15", "LG16", "LG17", "LG18", "LG19", "LG20", "LG21", "LG22", "LG23", "LG24", "LG25", "LG26", "LG27", "LG28", "LG28B", "LG29", "LG30", "LG30F", "LG31", "LG32", "LG33", "LG34", "LG35", "LG36", "LG36F", "LG37", "LG37M", "LG38", "LG39", "LG40", "LG41", "LG42", "LG42F", "LG43", "LG44", "LG44F", "LG45", "LG45M", "LG46", "LG47", "LG48", "LG48F", "LG49", "LG49B", "LG50"` for "carAur01.sm.fa"

Since other contigs of arAur01.sm.fa might be unclasped heterozygous sequences, we only keep chromosomes for the genome alignment. 15923983 is the line number of last chromosome sequence in the file of carAur01.sm.fa

```
head -15923983 carAur01.sm.fa > goldfish.fa
```

1.2.2 Get full-length CDS in fasta format

NOTE: please do NOT use CDS extracted using other software. Since AnchorWave filtered some CDS records to minimize the side effect of minimap2 limitation that "Minimap2 often misses small exons" (https://github.com/lh3/minimap2#limitations)
If you input files are compressed, please decompress them firstly.

```
anchorwave gff2seq -r Danio_rerio.GRCz11.dna.primary_assembly.fa -i Danio_rerio.GRCz11.102.chr.gff3 -o cds.fa
```

1.2.3 Map full-length CDS to reference and query genomes

-t is the number of thread for mapping, please adjust it according to your computational resource.
Empirically, the value of -k with 12 is a balance between mapping sensitivity and computational resource costing.

```
minimap2 -x splice -t 11 -k 12 -a -p 0.4 -N 20 Danio_rerio.GRCz11.dna.primary_assembly.fa cds.fa > ref.sam
minimap2 -x splice -t 11 -k 12 -a -p 0.4 -N 20 goldfish.fa cds.fa > carAur.sam
```

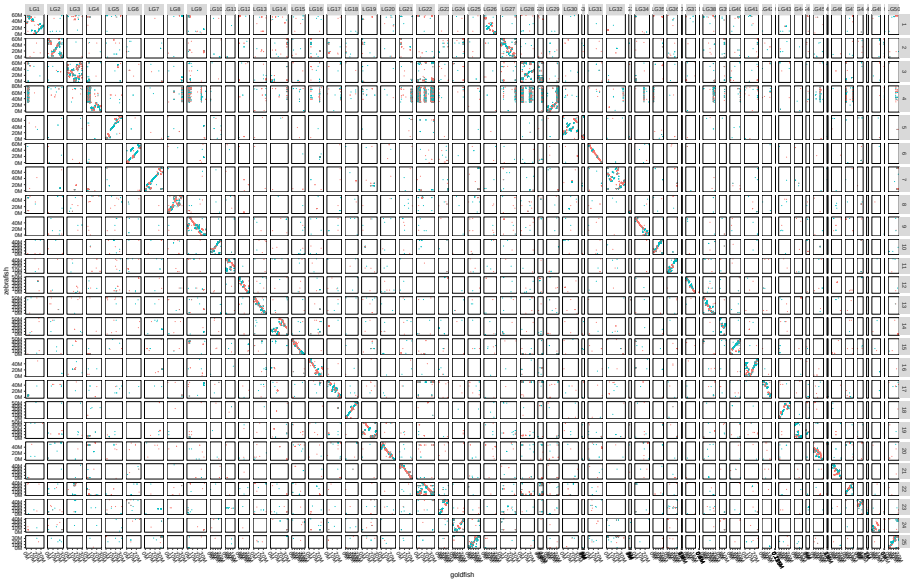
1.2.4 Visualize the full-length CDS mapping result

Transform sam file into a plain file. The scripts is distributed in the source code repository under ./scripts/alignmentToDotplot.pl

```
perl alignmentToDotplot.pl Danio_rerio.GRCz11.102.chr.gff3 carAur.sam > carAur.tab
```

Generate a plot

```
library(ggplot2)
library(compiler)
enableJIT(3)
## [1] 3
library(ggplot2)
library("Cairo")
changetoM <- function ( position ){
  position=position/1000000;
  paste(position, "M", sep="")
}
data =read.table("carAur.tab")
data = data[which(data$V1 %in% c("1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13", "14", "15", "16", "17", "18", "19", "20",
                                 "21", "22", "23", "24", "25", "MT" )),]
data = data[which(data$V3 %in% c("LG1","LG2","LG3", "LG4", "LG5", "LG6", "LG7", "LG8", "LG9", "LG10", "LG11", "LG12", "LG13", "LG14", "LG15",
                                 "LG16", "LG17", "LG18", "LG19", "LG20", "LG21", "LG22", "LG23", "LG24", "LG25", "LG26", "LG27", "LG28",
                                 "LG28B", "LG29",  "LG30", "LG30F", "LG31", "LG32", "LG33", "LG34", "LG35", "LG36", "LG36F", "LG37",
                                 "LG37M", "LG38", "LG39",  "LG40", "LG41", "LG42", "LG42F", "LG43", "LG44", "LG44F", "LG45",
                                 "LG45M", "LG46", "LG47", "LG48", "LG48F", "LG49", "LG49B", "LG50")),]
data$V1 = factor(data$V1, levels=c("1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13", "14", "15", "16", "17", "18", "19",
                                   "20", "21", "22", "23", "24", "25", "MT" ))
data$V3 = factor(data$V3, levels=c("LG1","LG2","LG3", "LG4", "LG5", "LG6", "LG7", "LG8", "LG9", "LG10", "LG11", "LG12", "LG13", "LG14", "LG15",
                                   "LG16", "LG17", "LG18", "LG19", "LG20", "LG21", "LG22", "LG23", "LG24", "LG25", "LG26", "LG27", "LG28",
                                   "LG28B", "LG29",  "LG30", "LG30F", "LG31", "LG32", "LG33", "LG34", "LG35", "LG36", "LG36F", "LG37",
                                   "LG37M", "LG38", "LG39",  "LG40", "LG41", "LG42", "LG42F", "LG43", "LG44", "LG44F",
                                   "LG45", "LG45M", "LG46", "LG47", "LG48", "LG48F", "LG49", "LG49B", "LG50" ))
ggplot(data=data, aes(x=V4, y=V2))+geom_point(size=0.5, aes(color=V5))+facet_grid(V1~V3, scales="free", space="free" )+ theme_grey(base_size = 120) +
    labs(x="goldfish", y="zebrafish")+scale_x_continuous(labels=changetoM) + scale_y_continuous(labels=changetoM) +
    theme(axis.line = element_blank(),
          panel.background = element_blank(),
          panel.border = element_rect(fill=NA,color="black", size=0.5, linetype="solid"),
          axis.text.y = element_text( colour = "black"),
          legend.position='none',
          axis.text.x = element_text(angle=300, hjust=0, vjust=1, colour = "black") )
```



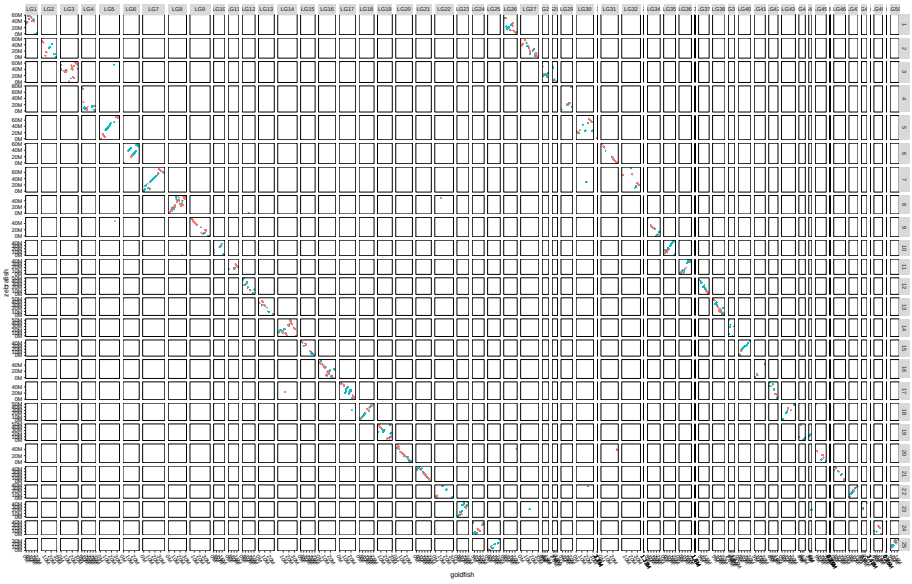
1.2.5 Using AnchorWave to identify collinear region and plot collinear anchors

By looking at the above plot, there are translocation variations and an unshared genome duplication, we use the `proali` function to identify collinear region.
We expect the genome alignment coverage on reference genome (zebrafish) equal with or smaller than 2. We expect the genome alignment coverage on query genome (goldsifh) equal with or smaller than 1. So we set parameter -R 2 -Q 1
As this stage we would like to identify collinear anchors and do not perform base-pair resolution sequence alignment. So we only set -n to output collinear anchors. Since novel anchors do not help to identify collinear blocks, by setting -ns , we do not identify novel anchors for plotting purpose.
This command could use up to a couple of tens Gb memory.

```
anchorwave proali -i Danio_rerio.GRCz11.102.chr.gff3 -r Danio_rerio.GRCz11.dna.primary_assembly.fa -a carAur.sam -as cds.fa -ar ref.sam -s goldfish.fa \
  -n align1.anchors -R 2 -Q 1 -ns

data =read.table("align1.anchors", head=TRUE)
data = data[which(data$refChr %in% c("1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13", "14", "15", "16", "17", "18", "19", "20",
                                     "21", "22", "23", "24", "25", "MT" )),]
data = data[which(data$queryChr %in% c("LG1","LG2","LG3", "LG4", "LG5", "LG6", "LG7", "LG8", "LG9", "LG10", "LG11", "LG12", "LG13", "LG14", "LG15",
                                       "LG16", "LG17", "LG18", "LG19", "LG20", "LG21", "LG22", "LG23", "LG24", "LG25", "LG26", "LG27", "LG28",
                                       "LG28B", "LG29",  "LG30", "LG30F", "LG31", "LG32", "LG33", "LG34", "LG35", "LG36", "LG36F", "LG37", "LG37M",
                                       "LG38", "LG39",  "LG40", "LG41", "LG42", "LG42F", "LG43", "LG44", "LG44F", "LG45", "LG45M", "LG46", "LG47",
                                       "LG48", "LG48F", "LG49", "LG49B", "LG50")),]
data$refChr = factor(data$refChr, levels=c("1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13", "14", "15", "16", "17", "18", "19",
                                           "20", "21", "22", "23", "24", "25", "MT" ))
data$queryChr = factor(data$queryChr, levels=c("LG1","LG2","LG3", "LG4", "LG5", "LG6", "LG7", "LG8", "LG9", "LG10", "LG11", "LG12", "LG13", "LG14",
                                               "LG15", "LG16", "LG17", "LG18", "LG19", "LG20", "LG21", "LG22", "LG23", "LG24", "LG25", "LG26",
                                               "LG27", "LG28", "LG28B", "LG29",  "LG30", "LG30F", "LG31", "LG32", "LG33", "LG34", "LG35", "LG36",
                                               "LG36F", "LG37", "LG37M", "LG38", "LG39",  "LG40", "LG41", "LG42", "LG42F", "LG43", "LG44", "LG44F",
                                               "LG45", "LG45M", "LG46", "LG47", "LG48", "LG48F", "LG49", "LG49B", "LG50" ))

ggplot(data=data, aes(x=queryStart, y=referenceStart))+geom_point(size=0.5, aes(color=strand))+
    facet_grid(refChr~queryChr, scales="free", space="free" )+ theme_grey(base_size = 120) +
    labs(x="goldfish", y="zebrafish")+scale_x_continuous(labels=changetoM) + scale_y_continuous(labels=changetoM) +
    theme(axis.line = element_blank(),
          panel.background = element_blank(),
          panel.border = element_rect(fill=NA,color="black", size=0.5, linetype="solid"),
          axis.text.y = element_text( colour = "black"),
          legend.position='none',
          axis.text.x = element_text(angle=300, hjust=0, vjust=1, colour = "black") )
```



1.3 Perform genome alignment

Here we set scoring parameters -B, -O1, -E1 and -O2, to compare the alignment result of using different scoring parameters.
Using the wavefront alignment (WFA) algorithm, the match score is constantly set as 0. Empirically, we always set -E2 as -1.
We set -w, and -fa3 to minimum the usage of sliding window alignment approach and novel anchors. Those parameter would improve the alignment while cost memory than the default parameters.
With the following setting, each thread could cost as high as 50 Gb memory, and we ran this command on a computer with 512Gb memory available with 10 threads.

```
/usr/bin/time ./anchorwave/anchorwave proali -i Danio_rerio.GRCz11.102.chr.gff3 -r Danio_rerio.GRCz11.dna.primary_assembly.fa \
  -a carAur.sam -as cds.fa -ar ref.sam -s goldfish.fa -n align1.anchors -o align1.maf -t 10 -R 2 -Q 1 -B -2 -O1 -4 -E1 -2 -O2 -80 -E2 -1 \
  -f align1.f.maf -w 38000 -fa3 200000 > fishlog1 2>&1
/usr/bin/time anchorwave proali -i Danio_rerio.GRCz11.102.chr.gff3 -r Danio_rerio.GRCz11.dna.primary_assembly.fa \
  -a carAur.sam -as cds.fa -ar ref.sam -s goldfish.fa -n align2.anchors -o align2.maf -t 9 -R 2 -Q 1 -B -4 -O1 -4 -E1 -2 -O2 -80 -E2 -1 \
  -f align2.f.maf -w 38000 -fa3 200000 > fishlog2 2>&1
```

For the above alignment, for each anchor and inter-anchor interval alignment, AnchorWave firstly try to use the WFA algorithm to perform alignment. If WFA costs too much memory, then switch to the ksw2_extd2_sse function implemented in the KSW2 library (https://github.com/lh3/ksw2). Taking the same input sequence, both WFA and ksw2_extd2_sse perform standard dynamic programming sequence alignment and outputs nearly identical alignments. On average, WFA is faster than ksw2_extd2_sse and for highly diverse sequence ksw2_extd2_sse might use less memory than WFA. If the multiply of the reference interval sequence and query interval sequence is smaller than 38000 * 38000 * 30, the standard synamic programming sequence alignment of ksw2_extd2_sse would be used for sequence alignment.
If we define the length of the longer sequence out of reference interval sequence and query interval sequence as a, if a <= 38000 * 15, the banded sequence alignment approach implemented in ksw2_extd2_sse would be used. And the banded width is calculated as (38000*38000*30)/2/a.
Otherwise, the sliding window approach is used with a sliding windows size of 38000.
The aim of the above process is to speed up the computing, by using WFA as much as

possible. And by using WFA and ksw2_extd2_sse, we would like to increase the usage of approximation sequence alignments. By dynamicly adjusting the band width, we would like to use a band width as large as possible with limited memory.

2 If we know the collinearity and whole-genome duplication

2.1 Prepare data

```
wget ftp://ftp.ensemblgenomes.org/pub/plants/release-34/gff3/zea_mays/Zea_mays.AGPv4.34.gff3.gz
gunzip Zea_mays.AGPv4.34.gff3.gz
wget ftp://ftp.ensemblgenomes.org/pub/plants/release-34/fasta/zea_mays/dna/Zea_mays.AGPv4.dna.toplevel.fa.gz
gunzip Zea_mays.AGPv4.dna.toplevel.fa.gz
wget ftp://ftp.ensemblgenomes.org/pub/plants/release-49/fasta/sorghum_bicolor/dna/Sorghum_bicolor.Sorghum_bicolor_NCBIv3.dna.toplevel.fa.gz
gunzip Sorghum_bicolor.Sorghum_bicolor_NCBIv3.dna.toplevel.fa.gz
```

2.2 Perform analysis using our pipeline

```
anchorwave gff2seq -r Zea_mays.AGPv4.dna.toplevel.fa -i Zea_mays.AGPv4.34.gff3 -o cds.fa
minimap2 -x splice -t 6 -k 12 -a -p 0.4 -N 20 Sorghum_bicolor.Sorghum_bicolor_NCBIv3.dna.toplevel.fa cds.fa > cds.sam
minimap2 -x splice -t 6 -k 12 -a -p 0.4 -N 20 Zea_mays.AGPv4.dna.toplevel.fa cds.fa > ref.sam
/usr/bin/time anchorwave proali -i Zea_mays.AGPv4.34.gff3 -as cds.fa -r Zea_mays.AGPv4.dna.toplevel.fa -a cds.sam -ar ref.sam \
  -s Sorghum_bicolor.Sorghum_bicolor_NCBIv3.dna.toplevel.fa -n anchors -e 2 -R 1 -Q 2 \
  -o alignment.maf -f alignment.f.maf -w 38000 -fa3 200000 > log 2>&1
```

3 Genome masking

Genome masking is not expected to improve the performance of AnchorWave.
AnchorWave do not utilize any soft masking information. Hard masking would increase the computational cost of AnchorWave.