Ben Iovino

12/2/2022

INFO519 – Introduction to Bioinformatics

Project Assignment

GitHub Repository: https://github.com/biovino1/genomeassembly

1) Basic Tasks

1. The github repository linked above has the five files needed to reproduce the results found in this project. The biovino_sing.def file was used to build the singularity image file necessary for this project, which was subsequently uploaded to the singularity sylabs website so that it can be remotely pulled instead of being built locally. The run_assembly.txt file loads the singularity module, pulls the .sif image file from sylabs, and executes biovino_record.txt from the singularity container. The first command in run_assembly.txt can be commented out to run this project on a local machine with singularity already installed. The biovino_record.txt file sets up the directory paths and runs all necessary commands to get the data from ncbi, find the right contig, and perform several different read simulations and assemblies. There are two SOAPdenovo2 config files - biovino_pe.config for paired end assembly, and biovino_se.config for single end assembly. The readme file on github also explains how to replicate this project and what each file accomplishes.
2. The software collected for this project is all in the biovino_sing.def file.
    1. Installed from apt: bc, zip, unzip, wget, tzdata, curl, build-essential, git, cmake, zlib1g, zlib1g-dev
    2. Cloned from github: wgsim, SOAPdenovo2
    3. Binary files: seqkit, ncbi-blast
3. The biovino_sing.def file can be used to build the biovino.sif singularity image that holds all the necessary software for this project. The image file is also stored on sylabs so that it can be remotely pulled from their servers. The biovino_record.txt is the script with all of the commands run to get the data in this project.
4. Data acquisition:
    1. Locust Genome taken from "https://api.ncbi.nlm.nih.gov/datasets/v1/genome/accession/GCA_000516895.1/download?include_annotation_type=GENOME_GFF,RNA_FASTA,CDS_FASTA,PROT_FASTA&filename=GCA_000516895.1.zip" -H "Accept: application/zip"
    2. There were 29 contigs with lengths equal to or greater than 80,000 nucleotides, which were isolated using an assortment of seqkit and linux commands

c. Under the student roster in Canvas, Iovino is the 7th student, therefore the 7th longest contig was pulled from the locust genome. This contig is about 93,000 base pairs long.

5. Explore assembly tools:

a. The default wgsim parameters are used as a baseline to compare the change of parameters to (number of reads was changed to decrease run time):

i. Default - wgsim -e 0.02 -N 10000 -1 70 -2 70 -r 0.001

ii. High Error - wgsim **-e 0.1** -N 10000 -1 70 -2 70 **-r 0.01**

iii. Low Reads - wgsim -e 0.02 -**N 1000** -1 70 -2 70 -r 0.001

iv. Short Reads - wgsim -e 0.02 -N 10000 **-1 30 -2 30** -r 0.001

v. Single End Reads - wgsim 0.02 -N 10000 -1 70 -2 70 -r 0.001

b. SOAPdenovo was used to assemble the reads from each of the wgsim runs above. There are two different config files used, one for paired end reads (biovino_pe.config) and one for single end reads (biovino_se.config). The only parameter that changes between these two files is at the end, where only one read is used for assembly (q) as opposed to two (q1, q2).

c. Five different assemblies were performed: Default (i), High Error (ii), Low Reads (iii), Short Reads (iv), and Single End Reads (v). The first four assemblies were created with paired end reads, the fifth one was created with single end reads.

Contigs	**Default**	**High Error**	**Low Reads**	**Short Reads**	**Single End**
Number	104	66	47	10	263
Mean Size	839	119	133	124	303
Median Size	365	115	124	108	237
Contig>1K	32	0	0	0	6
N50 (len, #)	1608, 17	117, 29	129, 20	122, 5	372, 69

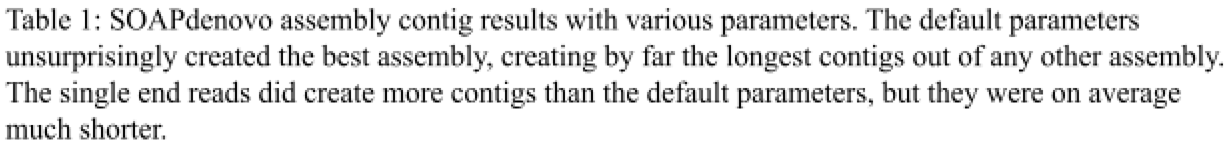


Table 1: SOAPdenovo assembly contig results with various parameters. The default parameters unsurprisingly created the best assembly, creating by far the longest contigs out of any other assembly. The single end reads did create more contigs than the default parameters, but they were on average much shorter.

Scaffolds	Default	High Error	Low Reads	Short Reads	Single End
Number	11	66	46	9	263
Mean Size	8486	119	174	275	303
Median Size	245	115	123	134	237
Scaffolds>1K	3	0	0	0	6
N50 (len, #)	38461, 2	117, 29	144, 13	574, 3	372, 69

Table 2: SOAPdenovo assembly scaffold results with various parameters. The same results can essentially be seen from Table 1, but it is important to look at how the contigs were used to make scaffolds. The default parameters made for the best scaffolds, with a mean size of almost 100x that of every other run.

An individual table was made for contigs and scaffolds to see the connection between the two. Contigs are generated from reads, and scaffolds are generated from contigs. One would expect longer and more accurate reads to be assembled into longer contigs and then into longer scaffolds because there are more matching segments. This is demonstrated in Table 1 where an error rate of 2%, mutation rate of 0.1%, read lengths of 70 bp, and read number of 10,000 generated the longest contigs. The N50 contig had a length of 1608 base pairs, and 17 contigs were this length or greater. No other assembly comes close to this N50 value. The ‘single end’ read assembly does have a greater amount of contigs than the ‘default’ assembly, 263 contigs vs. 104 contigs, but the N50 is much lower at 372 and the median contig size is over 100 bp shorter. The ‘high error’, ‘low reads’, and ‘short reads’ assemblies all have N50 values of 115-130, significantly smaller than of the ‘default’ assembly, meaning they have significantly smaller contigs.

It is obvious from Table 2 that the longer contigs you have, the longer scaffolds you have. Longer scaffolds are desirable because they represent a more completely assembled genome, which is the goal of genome assembly. The ‘default’ assembly outperforms every other assembly by an extremely wide margin, with the N50 contig being 38461 bp, and there are two scaffolds this length or greater. The original locust contig itself is about 93,000 bp’s long, so these two scaffolds make up almost 83% of the genome, which is good coverage for only two scaffolds. The N50 values for every other assembly essentially remains the same because every other assembly fails to generate scaffolds from the contigs. If you compare the number of contigs vs. the number of scaffolds, they are exactly the same for the ‘high error’ and ‘single read’ assemblies, meaning all of the scaffolds are just contigs that were unable to be joined together. The ‘low reads’ and ‘short reads’ assemblies have one less scaffold than contig, meaning only two contigs were joined together for each assembly. This is still a very poor assembly performance.

2) Advanced Tasks

1. The 7th contig from the Locust genome was used to predict genes using the Augustus web server, https://bioinf.uni-greifswald.de/webaugustus/prediction/create. There is no species parameter for Locusta Migratory, so genes were predicted using three different species parameters to see which one predicted the most genes: *Drosophila melanogaster* (fruit fly), *Heliconius melpomene* (butterfly), and *Tribolium castaneum* (red flour beetle).
    a. *D. melanogaster* - 2 genes predicted, both with several introns and coding sequences
    b. *H. melpomene* - 18 genes predicted, most only have a few introns and coding sequences and a couple with many introns and coding sequences
    c. *T. castaneum* - 22 genes predicted, tend to be of medium-long length
2. The 22 predicted genes from the *T. castaneum* species parameter were blasted against proteins using NCBI's blastp web program, hoping there would be more blast hits. The 16th gene showed good hits, with ten E-values below 1e-30 and query coverage of higher than 80%. The three promising candidates taken from these results were transposase proteins, two from different *S. enterica* subspecies and one from *T. clavipes* (orb weaver spider). All 100 results from blastp were transposase proteins, giving credence to the 16th predicted gene model from *T. castaneum* from the 7th contig being a transposase.
3. Phylogenetic analysis was performed using MEGA. An alignment was made using the MUSCLE algorithm, the pairwise distances were calculated between the four sequences (gene 16 and the three transposase proteins), and a neighbor joining tree was generated.

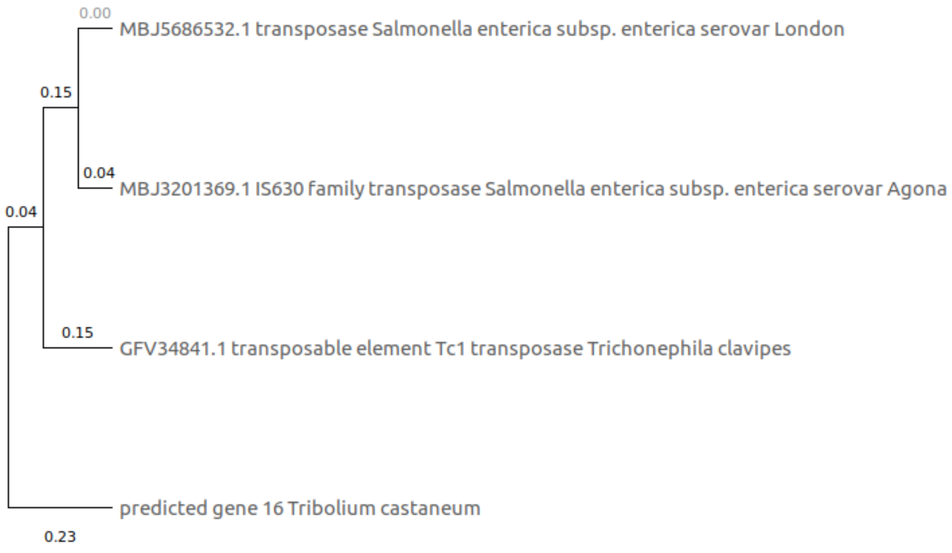

0.00
MBJ5686532.1 transposase Salmonella enterica subsp. enterica serovar London
0.15
0.04
MBJ3201369.1 IS630 family transposase Salmonella enterica subsp. enterica serovar Agona
0.04
0.15
GFV34841.1 transposable element Tc1 transposase Trichonephila clavipes
predicted gene 16 Tribolium castaneum
0.23

Figure 1: Neighbor-Joining phylogenetic tree produced from a Muscle alignment of four protein sequences, all done in MEGA. Branch lengths are shown as the numbers on each branch. The first three sequences are from blastp results on the fourth sequence, a predicted gene from Augustus.

The above neighbor-joining tree shows the two *S. enterica* species as very close to each other (0.04), then *T. clavipes* 0.3 from their ancestral nodes, and then the predicted gene 0.42 away from *T. clavipes* and 0.42 away from the ancestral node of the *S. Enterica* species. The three protein sequences from blastp were chosen to see if the predicted gene was closer to any specific species's transposase, but the results from this tree simply show the predicted gene is the farthest one away from all other sequences.

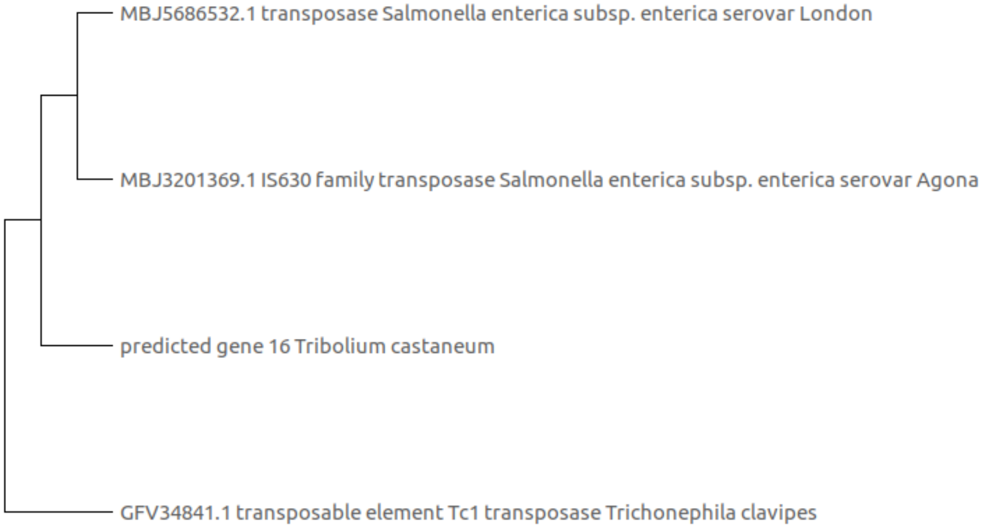
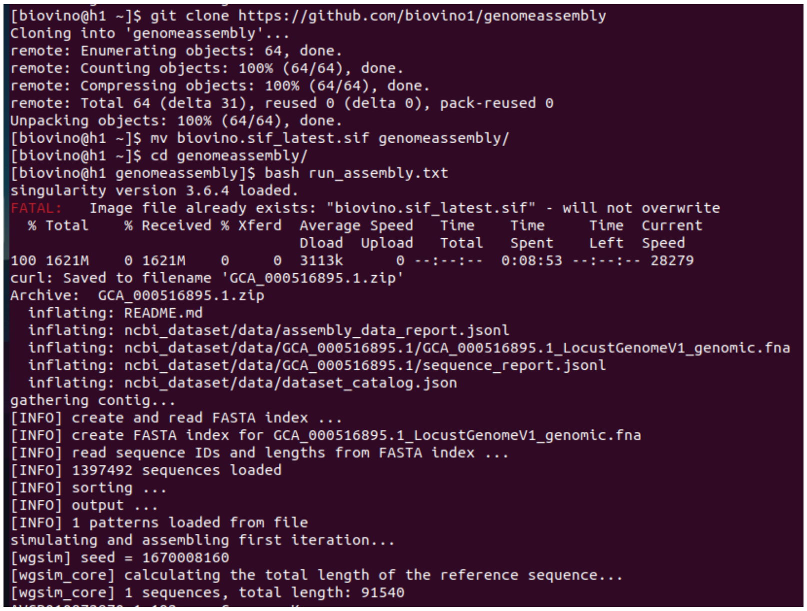

MBJ5686532.1 transposase Salmonella enterica subsp. enterica serovar London
MBJ3201369.1 IS630 family transposase Salmonella enterica subsp. enterica serovar Agona
predicted gene 16 Tribolium castaneum
GFV34841.1 transposable element Tc1 transposase Trichonephila clavipes

Figure 2: Maximum-Parsimony phylogenetic tree produced from a Muscle alignment of four protein sequences, all done in MEGA. The first, second, and fourth sequences are blastp results from the third sequence, a predicted gene from Augustus.

The above maximum-parsimony tree still shows the two *S. enterica* species as the closest, but unlike the neighbor-joining tree, the predicted gene from Augustus is closer to the *S. enterica* species than the *T. clavipes* protein, which is now the farthest sequence away. Without distances, however, it cannot be determined exactly how far each of the nodes and branches are from each other. One explanation for the difference in these trees is perhaps the length of the predicted gene. It is much shorter than the other three sequences and requires less insertions or deletions to align with the *S. enterica* species than *T. clavipes*, which is a much longer sequence (over twice the length of the predicted gene) and may require more character state changes, i.e. mutations, which the maximum parsimony tree does not reward.

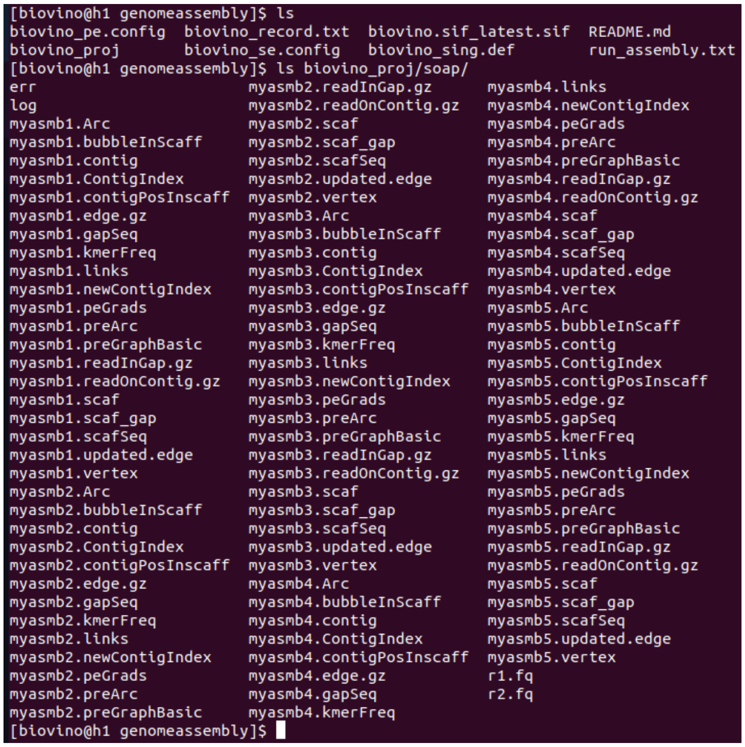
3) Optional Task

1. All files needed to replicate the results from the basic tasks can be found in this github repository: https://github.com/biovino1/genomeassembly. To run this on Carbonate, all that needs to be done is to execute the run_assembly.txt file. This will load the singularity module, download a pre-built singularity image file from sylabs (which was built from biovino_sing.def), and then execute the biovino_record.txt file while in the singularity container. Evidence of this working is provided by the screenshots below. A slurm job file is also included in the repository so that it can be submitted to the job manager.



```
[biovino@h1 ~]$ git clone https://github.com/biovino1/genomeassembly
Cloning into 'genomeassembly'...
remote: Enumerating objects: 64, done.
remote: Counting objects: 100% (64/64), done.
remote: Compressing objects: 100% (64/64), done.
remote: Total 64 (delta 31), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (64/64), done.
[biovino@h1 ~]$ mv biovino.sif_latest.sif genomeassembly/
[biovino@h1 ~]$ cd genomeassembly/
[biovino@h1 genomeassembly]$ bash run_assembly.txt
singularity version 3.6.4 loaded.
FATAL:   Image file already exists: "biovino.sif_latest.sif" - will not overwrite
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 1621M    0 1621M    0     0  3113k      0 --:--:--  0:08:53 --:--:-- 28279
curl: Saved to filename 'GCA_000516895.1.zip'
Archive:  GCA_000516895.1.zip
  inflating: README.md
  inflating: ncbi_dataset/data/assembly_data_report.jsonl
  inflating: ncbi_dataset/data/GCA_000516895.1/GCA_000516895.1_LocustGenomeV1_genomic.fna
  inflating: ncbi_dataset/data/GCA_000516895.1/sequence_report.jsonl
  inflating: ncbi_dataset/data/dataset_catalog.json
gathering contig...
[INFO] create and read FASTA index ...
[INFO] create FASTA index for GCA_000516895.1_LocustGenomeV1_genomic.fna
[INFO] read sequence IDs and lengths from FASTA index ...
[INFO] 1397492 sequences loaded
[INFO] sorting ...
[INFO] output ...
[INFO] 1 patterns loaded from file
simulating and assembling first iteration...
[wgsim] seed = 1670008160
[wgsim_core] calculating the total length of the reference sequence...
[wgsim_core] 1 sequences, total length: 91540
```

Figure 3: Screenshot showing the github repository being cloned and run_assembly.txt being executed. The singularity image file was already located in the directory (saved from a previous run of this project as to save time) so it was not overwritten from a remote pull. Curl gathered the genome assembly data from ncbi. Seqkit sorted the contigs as to gather the 7th longest one. Wgsim performs the first read simulation.



```
[biovino@h1 genomeassembly]$ ls
biovino_pe.config  biovino_record.txt  biovino.sif_latest.sif  README.md
biovino_proj       biovino_se.config   biovino_sing.def        run_assembly.txt
[biovino@h1 genomeassembly]$ ls biovino_proj/soap/
err                       myasmb2.readInGap.gz      myasmb4.links
log                       myasmb2.readOnContig.gz   myasmb4.newContigIndex
myasmb1.Arc               myasmb2.scaf              myasmb4.peGrads
myasmb1.bubbleInScaff     myasmb2.scaf_gap          myasmb4.preArc
myasmb1.contig            myasmb2.scafSeq           myasmb4.preGraphBasic
myasmb1.ContigIndex       myasmb2.updated.edge      myasmb4.readInGap.gz
myasmb1.contigPosInscaff  myasmb2.vertex            myasmb4.readOnContig.gz
myasmb1.edge.gz           myasmb3.Arc               myasmb4.scaf
myasmb1.gapSeq            myasmb3.bubbleInScaff     myasmb4.scaf_gap
myasmb1.kmerFreq          myasmb3.contig            myasmb4.scafSeq
myasmb1.links             myasmb3.ContigIndex       myasmb4.updated.edge
myasmb1.newContigIndex    myasmb3.contigPosInscaff  myasmb4.vertex
myasmb1.peGrads           myasmb3.edge.gz           myasmb5.Arc
myasmb1.preArc            myasmb3.gapSeq            myasmb5.bubbleInScaff
myasmb1.preGraphBasic     myasmb3.kmerFreq          myasmb5.contig
myasmb1.readInGap.gz      myasmb3.links             myasmb5.ContigIndex
myasmb1.readOnContig.gz   myasmb3.newContigIndex    myasmb5.contigPosInscaff
myasmb1.scaf              myasmb3.peGrads           myasmb5.edge.gz
myasmb1.scaf_gap          myasmb3.preArc            myasmb5.gapSeq
myasmb1.scafSeq           myasmb3.preGraphBasic     myasmb5.kmerFreq
myasmb1.updated.edge      myasmb3.readInGap.gz      myasmb5.links
myasmb1.vertex            myasmb3.readOnContig.gz   myasmb5.newContigIndex
myasmb2.Arc               myasmb3.scaf              myasmb5.peGrads
myasmb2.bubbleInScaff     myasmb3.scaf_gap          myasmb5.preArc
myasmb2.contig            myasmb3.scafSeq           myasmb5.preGraphBasic
myasmb2.ContigIndex       myasmb3.updated.edge      myasmb5.readInGap.gz
myasmb2.contigPosInscaff  myasmb3.vertex            myasmb5.readOnContig.gz
myasmb2.edge.gz           myasmb4.Arc               myasmb5.scaf
myasmb2.gapSeq            myasmb4.bubbleInScaff     myasmb5.scaf_gap
myasmb2.kmerFreq          myasmb4.contig            myasmb5.scafSeq
myasmb2.links             myasmb4.ContigIndex       myasmb5.updated.edge
myasmb2.newContigIndex    myasmb4.contigPosInscaff  myasmb5.vertex
myasmb2.peGrads           myasmb4.edge.gz           r1.fq
myasmb2.preArc            myasmb4.gapSeq            r2.fq
myasmb2.preGraphBasic     myasmb4.kmerFreq
[biovino@h1 genomeassembly]$ █
```

Figure 4. Screenshot showing all of the SOAPdenovo results in the corresponding directory.

2. Secondary structure prediction was performed on the predicted *T. castaneum* gene 16 using Jpred4. The consensus structure prediction, taken from several different prediction methods, is given by the 'jnetpred' line in the middle of Figure 5. Alpha helices are marked as red tubes, of which there are 4 predicted, and beta sheets are marked as green arrows, of which there are 3 predicted. The string of numbers from the jnetpred predictions range from 0-9, zero being an exposed residue and nine being a buried residue. The predicted structure has a mix of both, although it appears to have long runs of hydrophobic regions (15-25, 85-95). Confidence values are given by the black bars, and the hydrophobic regions appear to have the most confidence, indicating that this likely transposase is more hydrophobic than hydrophilic.

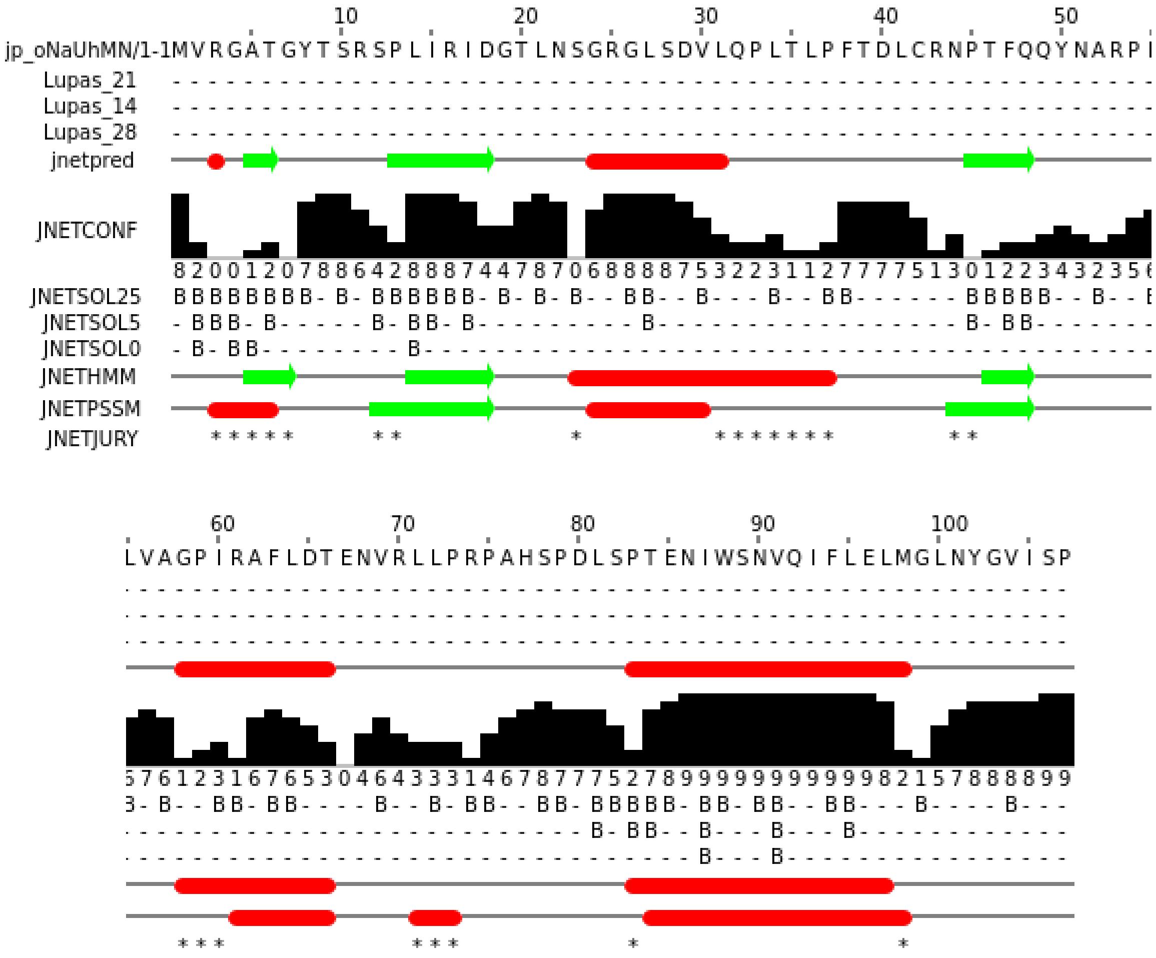


Figure 5: Jpred4 secondary structure prediction results on the *T. castaneum* predicted gene. The figure was split into two different pictures as to make it more visible. The first half (residues 0-55) is on top and the second half (residues 55-107) are on the bottom. The consensus prediction is in the middle of the pictures - red is consensus alpha helices, green is consensus beta sheets.