Institute of Infection and Global Health

Introduction to sequence informatics

Transcriptomics - Quantification and Differential Expression

During session 2, we introduced the idea of using transcriptomic data to identify genes by mapping reads back to an unannotated genome. This approach worked because reads from a transcriptomic dataset originate from only a small portion of the genome. Therefore, any regions where we saw a pileup of reads could be marked as a transcribed feature, the most common of which is a gene.

The majority of transcriptomic studies, however, are focused not on identifying where genes are but calculating the extent to which they are being transcribed. In a focused study where you have a single gene of interest, this could be achieved by taking the sample, reverse transcribing the RNA into DNA and then using specific primers to quantify the level of the individual transcript. RNA-Seq, on the other hand, leverages the high throughput of next-generation sequencers to sequence all of the transcripts at once, allowing for the generation of expression profiles across the entire genome.

RNA-Seq allows us to determine the expression profile for each sample and compare the patterns between sample groups to identify genes which are differentially expressed. As we are able to examine the entire genome this approach allows us to identify changes in genes without a priori knowledge of their role in a given biological process. Traditionally RNA-Seq has used mapping approaches similar to those you have already encountered to identify where reads map to a genome or set of transcripts. The major difference is the inclusion of split alignment, whereby two parts of a read may be split with a small gap. This type of alignment is required due to the fact that eukaryotic genomes separate some genes into multiple exons. Split read alignment allows us to identify sequences that span exon-intron-exon boundaries and subsequently position part of the read to one exon and part of it to the next exon.

Recent approaches have moved away from full alignment to using pseudo-alignment or k-mer counting. Software such as Kallisto and Salmon work by quickly identifying the most likely mapping location for a read without performing the full mapping. They then directly output counts of how many reads belong to each transcript. This significantly increases the speed at which they can process data, however, they do not retain alignment data making it harder to manually examine your results. Both of these approaches rely upon a set of transcripts. This may be in the form of an annotated reference genome or it may be generated de novo using software such as Trinity. During this session, we will use reads generated from the livestock pathogen *T. congolense* comparing the genes expressed during experimental mouse and natural fly infections.

During the exercises you will finish processing the final sample and examine it in Artemis. You will then generate read counts for each transcript and analyse variation in the expression profiles using R.

Exercise 1

During this exercise we will map our reads against the *T. congolense* reference genome, examine the resulting BAM file and then generate a list of read counts.

1. Create a new directory called session5 and enter it.

2. During exercise 1 we are going to map the reads for a single sample to our reference using a piece of software called hisat2. The reference file is currently located at:

   /working/Informatics/session5/IL3000g.fa

   Copy it into your session5 directory.

3. As always, the next step is to index the reference, the command for this is:

   hisat2-build IL3000g.fa IL3000

   This indexing step may take a couple of minutes.

4. During the previous sessions you have copied all of the data into your own home directory. For this session we will be using data stored in a different directory from where you will be working.

   The files for this session can be found in: /archived/Informatics/session5

   You will need to refer to this location when inputting your commands but you will be saving your output into the session5 directory you have just created. This approach, of storing raw data in one location and working from another, is fairly typical. One of the main uses here is that it maintains a separation between the raw data and the working data. If we were to accidentally delete the latter we could easily rerun the commands that created it, if we did that to the former then we might have to go back to the lab and rerun the entire experiment from the start!

5. We are now going to map the reads for sample HT6 to the reference. The command for is below and should be typed as a single line. Be careful with where you put the spaces.

   hisat2 -x IL3000 -p 1 —max-intronlen 1000 -1 /archived/Informatics/session5/A.1.gz -2 /archived/Informatics/session5/A.2.gz | samtools view -bS - > A.bam

   This will take around 10 minutes to run. Note that we have to tell hisat2 the full location of our two reads as they are not in the current directory. The output, however, will be saved to this directory.

6. Sort and index your bam file using the following two commands:

   samtools sort -@ 1 A.bam -o sorted.A.bam
   samtools index sorted.A.bam

7. Delete A.bam

8. Now that we have mapped and sorted our reads we need to examine the mapping using Artemis. Open Artemis. Make sure you tell it to load IL3000g.fa as part of the command.

9. You should now see the familiar panels and list of potential stop codons. We need to load a couple of additional files to check our mapping. First we'll load the annotation for this genome, which is stored in the Tc.embl file. Load it in using

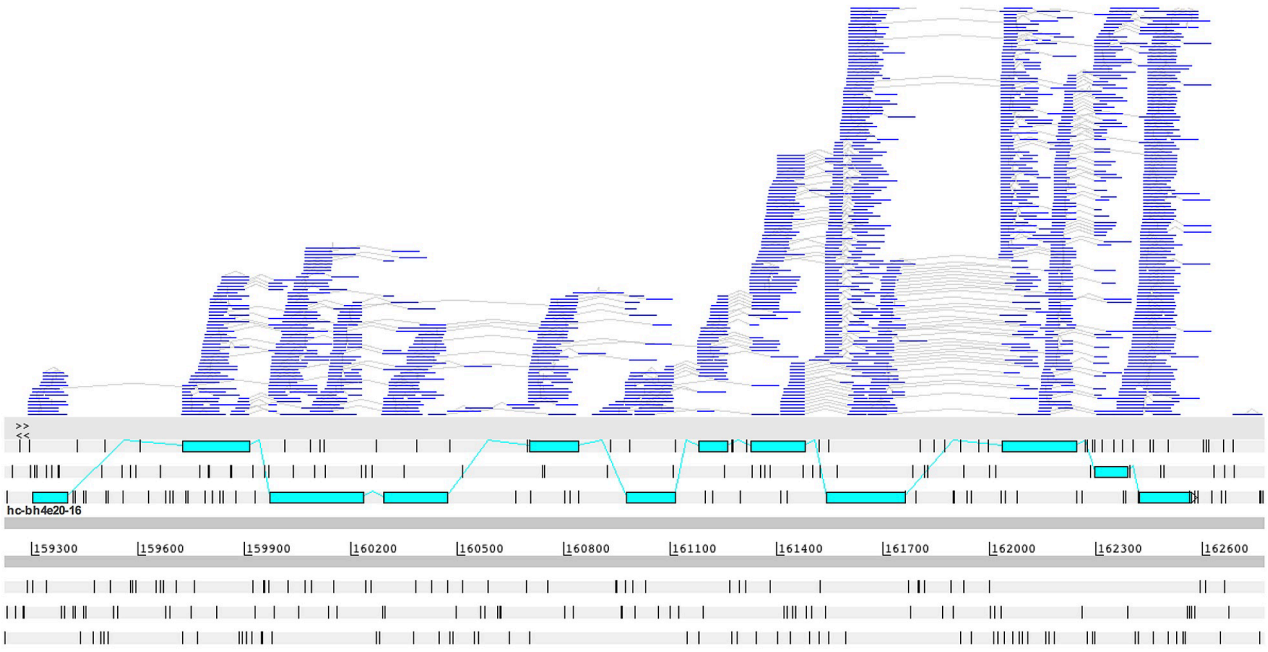
   File -> Read an entry into -> IL300g.fa

   Find and select Tc.embl. Once it has loaded turn off the stop codons.

10. Next we need to load our bam file using:

    File -> Read BAM / CRAM / VCF

    Select your sorted bam file, load it into artemis and spend a few minutes examining the mapping. You should see that there are clusters of reads. How would you check if these were mapping to genes? (Hint: You did this during an earlier session)



11. After you have spent some time examining the mapping exit Artemis. One of the primary uses of RNA-Seq is to calculate the expression profiles of genes and compare the patterns in one sample to those in another. To do this we need to take a list of genes and calculate the number of reads that map to it. Depending upon the software which you use we may normalise the value. Here we are going to use a piece of software called featureCounts and a second annotation file that is in the gtf format. FeatureCounts will use this annotation file to scan through each of the bam files, count the number of reads mapping to each gene and then output a single file of the raw read counts. This number is unnormalised, which means that we do not correct for the size of the gene or the total number of reads in a given sample. Normalisation is often an important step in the analysis of RNA-Seq data, as it allows us to more easily compare genes and samples.

    For example, imagine we had two genes, A and B, which are 7kb long and 14kb long and that are expressed at the same frequency. So for every mRNA originating from gene A there is one from gene B.

    After running our sequencing we have a total of 15,000 reads - How many came from each gene?

    The answer is 5,000 from gene A and 10,000 from gene B gene B is twice the size of gene A. If we had simply counted the total number of reads we might conclude that gene B was expressed at a higher level than gene A.

    During normalisation we correct for this by dividing the number of reads by the length of the gene to get the Reads Per Kilobase, for example:

    Gene A: 5,000 reads / 5kb = 1,000 reads per kb (RPK)
    Gene B: 10,000 reads / 10kb = 1,000 reads per kb (RPK)

    The second stage of normalisation is to correct for the total sequencing depth. We saw last session the effect of over sequencing a small genome - everything was completely covered by reads. But what if we had a second sample where we had only sequenced every gene once? The reads per kb would look significantly lower for each gene even if they were expressed in the same proportions. To correct for this, we divide each read per kilobase value by the total number of reads and then divide that by 1,000,000. This is the per million scaling factor and gives us a final normalised value known as the TPM.

    Gene A has an RPK value of 1,000 in sample 1 and 3,000 in sample 2. The total number number of reads in sample 1 is 3,000,000, while the total number in sample 2 is 10,000,000.

    What is the final TPM value for each gene?

    TPM normalisation is just one of many approaches to normalisation. You may see references to RPKM and FKPM in the literature, these use a similar method but perform the calculations in a slightly different order.

12. Before we can normalise any values we need to count how many reads map to each gene. To do this we will use a program call featureCounts. We also need to know the location of every gene, which we have stored in a GTF file called IL3000.gtf.

13. Copy IL3000.gtf from the /archived/Informatics/session5 folder to your current location.

14. We will now run featureCounts on the sample you have just mapped using the following command:

    featureCounts -T 1 -p -t exon -g gene_id -M -a IL3000.gtf -o TcA_counts sorted.A.bam

    This will produce two output files, examine them both using less and then delete them.

15. The reason we deleted the two output files is because we need to run featureCounts on all of the samples at once. You might be wondering what other samples there are - If you look in /archived/Informatics/session5 you should see a number of bam files. We didn't copy them to your home directory because that would require ~40gb of space each. Instead we are going to just tell featureCounts where they are located.

    featureCounts -T 1 -p -t exon -g gene_id -M -a Il3000.gtf -o Tc_counts sorted.A.bam /archived/Informatics/session5/*bam

Exercise 2

The comparison of transcription profiles is known as differential expression analysis. This analysis involves collecting together the read count data for every transcript, then comparing the expression levels between groups. Each group of samples represents a single experimental condition, with each group containing a minimum of 3 replicates (though ideally we would have more).

We are going to perform our analysis in R using a package called DESeq2, which has been developed specifically for the analysis of RNA-Seq data. We will load in the output from featureCounts (Tc_counts) and then use it to determine which genes display patterns of differential expression when comparing each group.

For this experiment we have two groups of samples - Samples identified with a number came from a study in Edinburgh where mice were experimentally infections with T. congolense. We are going to compare them with the second group of samples, which are identified with by a letter, which come from naturally infected tsetse flies from Kenya.

1. Double check that you have created Tc_counts and then load R

2. You will need to install and then load the DESeq2 package into R using one of the commands that you learned during the previous sessions.

3. Load your Tc_counts file into R using the command

   countdata <- read.table("Tc_counts", header=T, row.names=1, sep="\t")

4. Examine the countdata variable. The row names of countdata represent the gene IDs while the column names include annotation information and the sample names. It should be clear from a quick glance that the column names for the individual samples are rather unwieldy.

   Using the colnames command and what you have already learned rename the sample names to either mouse_ID or tsetse_ID. For example change

   X.archived.Informatics.session5.sorted.B.bam

   To

   tsetse_B

   Do this for each of the samples in countdata.

5. It will be easier if all of the sample columns are grouped by type but tsetse_A is out of position. Create a new variable called **countdata2** that contains only the sample columns and reorder it to group tsetse_A with the other tsetse samples.

6. We are also going to remove any genes where the total number of reads mapped across all of the samples is below 10. When the total level of expression is this low it is impossible for a gene to be differentially expressed. Removing these genes has two benefits. Firstly, a smaller dataset can be analysed fasted and secondly, it reduces the false discovery rate due to multiple testing.

   Remove the reads where the total counts are below 10 using the following two commands:

   totalcounts <- apply(countdata2, 1, sum)

   countdata2 <- countdata2[totalcounts > 10,]

   How many genes have you removed by performing this step?

   What percentage of all genes does this represent?

7. We will need to provide DESeq2 the details of which samples are which. We will create a small data frame listing each sample and whether it is a tsetse (T) or mouse (m) sample. The commands are:

   condition <- factor(c(rep("t", 6), rep("m", 6)))

   coldata=data.frame(row.names=colnames(countdata2), condition)

8. Look at the two variables that you have just created - do they make sense? Is every sample in the correct group?

9. Now that we have set our input variables we can perform the differential expression calculations. The first of these two commands will collect the information from countdata2, coldata and condition into a DESeq2 object, the second will then perform the calculations.

   dds <- DESeqDataSetFromMatrix(countData=countdata2, colData=coldata, design=~condition)

   dds <- DESeq(dds)

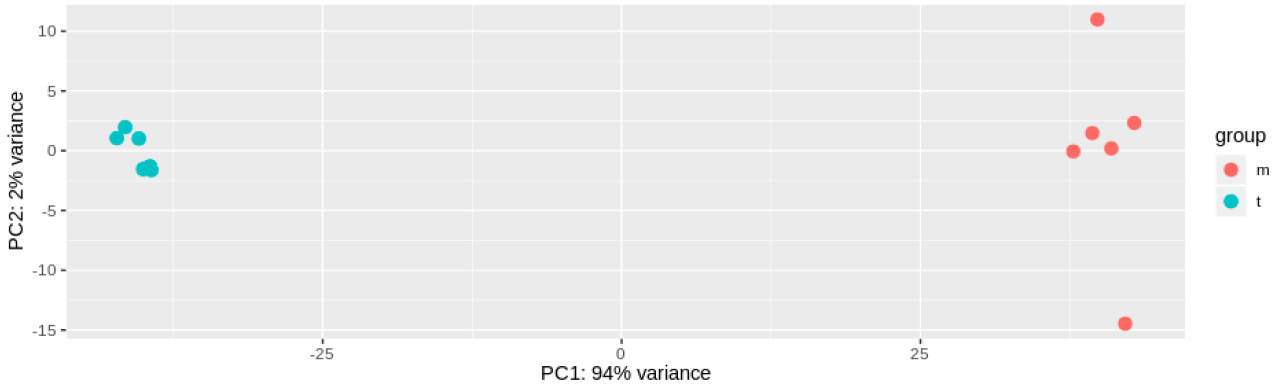
10. For our first analysis we are going to look at how well each sample clusters with others from the same group using a PCA plot. First we need to log transform the data using the command

    rlg <- rlog(dds)

    We will then create our PCA plot using

    plotPCA(rlg)

    Which should look like this



    PCA plots are one of the most common methods of looking for groups within your data as they take all of the samples, compare them together and then plot them based on the patterns within the data. In our dataset we see two clear clusters and each of them is comprised entirely of samples from mouse or from tsetse. We can also see that the variance on the x-axis is 94%. This means that 94% of ALL the differences in the data contribute to separating the two groups in this way. The variance on the y-axis is only 2% and most of this is actually from only 2 of the samples!

    This is what we would expect - it means that the parasites in tsetse are expressing genes in a very different pattern from the parasites in mice. In many experiments the differences are far subtler with groups overlapping or not clustering at all.

11. Imagine we had a third group of samples that came from natural cow infections.

    Where would you expect them to be located on your plot?

    There are two mouse samples that cluster away from the main group - what do you think could have caused this?

12. A second approach to assessing the overall similarity between samples is to plot them using a heatmap. For this we need to install and load two additional packages, gplots and RColorBrewer.

    install.packages("gplots")
    install.packages("RColorBrewer")
    library(gplots)
    library("RColorBrewer")

    This may ask if you wish to create a personal library - if it does enter yes at the prompt.

13. We then need to calculate the distances between the samples using

    sampleDists <- dist(t(assay(rlg)))

    sampleDistMatrix <- as.matrix(sampleDists)

14. Finally we create the heatmap using the following two commands

    colours = colorRampPalette(rev(brewer.pal(9, "Blues")))(255)
    heatmap.2(sampleDistMatrix, trace="none", col=colours)

    How does your heatmap compare to the PCA plot?

15. While it is important to check whether all the samples from a given experimental condition cluster together, the most important step in our analysis is to determine if a gene has significantly different expression patterns in one condition relative to another. Up until now we have been comparing all of the samples with each other, for the following steps we will compare only two conditions with one another.

    The instructions below will compare the control samples with those from the low temperature group. Once you have completed this analysis pick a second comparison (control vs high temperature or low temperature vs high temperature) and repeat the analysis to see how your results change.

    To compare our controls with the low temperature samples we first need to summarise the results using:

    resultsMT <- results(dds, contrast=c("condition", "m", "t"))
    resultsMT <- resultsMT[!is.na(resultsMT[,6]),]

16. Before we identify which genes have variable expression between our two samples we are first going to look at the overall patterns, using the following plot

    plotMA(resultsCLT)

    This command plots the mean expression for each gene over both conditions against the log change, which indicates if expression was higher in one condition relative to another. Positive values on the y-axis indicate higher expression in the first condition (here it is the mouse), negative values indicate higher expression in the second (tsetse). Genes with an adjusted p-value < 0.1 are marked in red.

    From this chart we can see that the majority of genes have a normalised expression level of over 100 but less than 10,000.

    Using the abline command construct a square around the main cluster of genes. What values did you need to use for the horizontal and vertical lines?

17. Examine the contents of resultsMT. The command

    mcols(resultsMT)

    will provide additional information about each of the columns. The two most important columns are log2FoldChange, which tells us how different the two conditions are and padj, the Benjamini–Hochberg adjusted p-value, which corrects for the fact that we have calculated pvalues for thousands of genes.

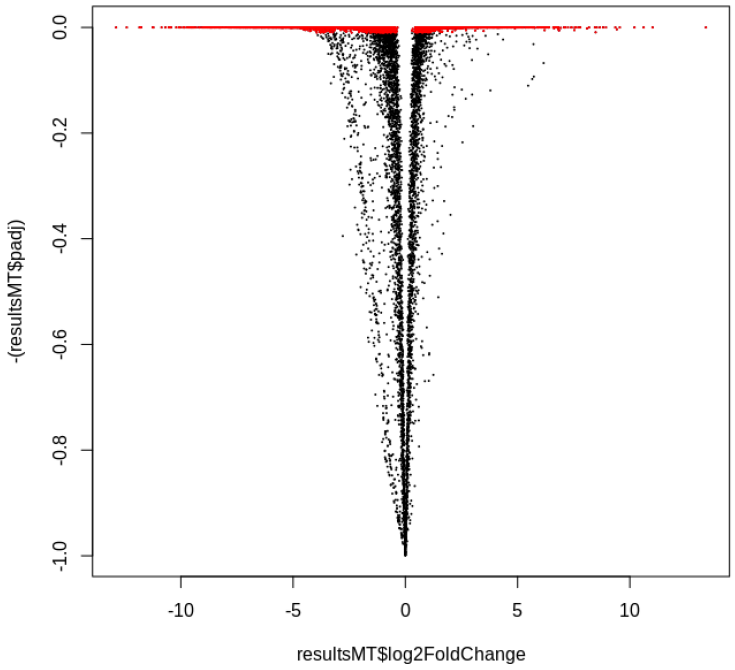
    A padj of 0.05 represents a False Discovery Rate of 5%. In other words if we set our significance threshold at padj < 0.05 then less than 5% of the genes in this list are false positives.

    How many genes have a padj below 0.05?

18. One of the strengths of R is that it allows us to quickly examine the data in more intuitive visual formats. DESeq2 provides a range of options for doing so but for now we are going to use the base plotting function of R to produce the volcano plot on the next page.

    This plot shows the distribution of the fold change between the tsetse and the mouse samples plotted against the adjusted p-values. A positive log2(fold change) indicates there was higher expression of the gene in the tsetse samples relative to the mouse samples, a negative value indicates the reverse. The plot was produced using the following commands:

    ourthreshold <- resultsMT[resultsMT$padj < 0.01,]
    plot(resultsMT$log2FoldChange, -(resultsMT$padj), pch=19, cex=0.1)
    points(ourthreshold$log2FoldChange, -(ourthreshold$padj),cex=0.1,pch=19, col="red")



Volcano plots such as these often log transform the adjusted p-values in addition to the fold change data. Adjust your commands to do so, you will need to add the log10 function to two of the above lines. It is more common to produce volcano plots that look like the figure below as it spreads out the points that are most likely to be significant.

Amend your commands to produce the plot below. You will need to change the significance threshold to 1e-10 to do so.