Unit 1. Introduction to computational genomics

Explains the historical context and how the sequencing data has increased (not important).

Genome Completeness

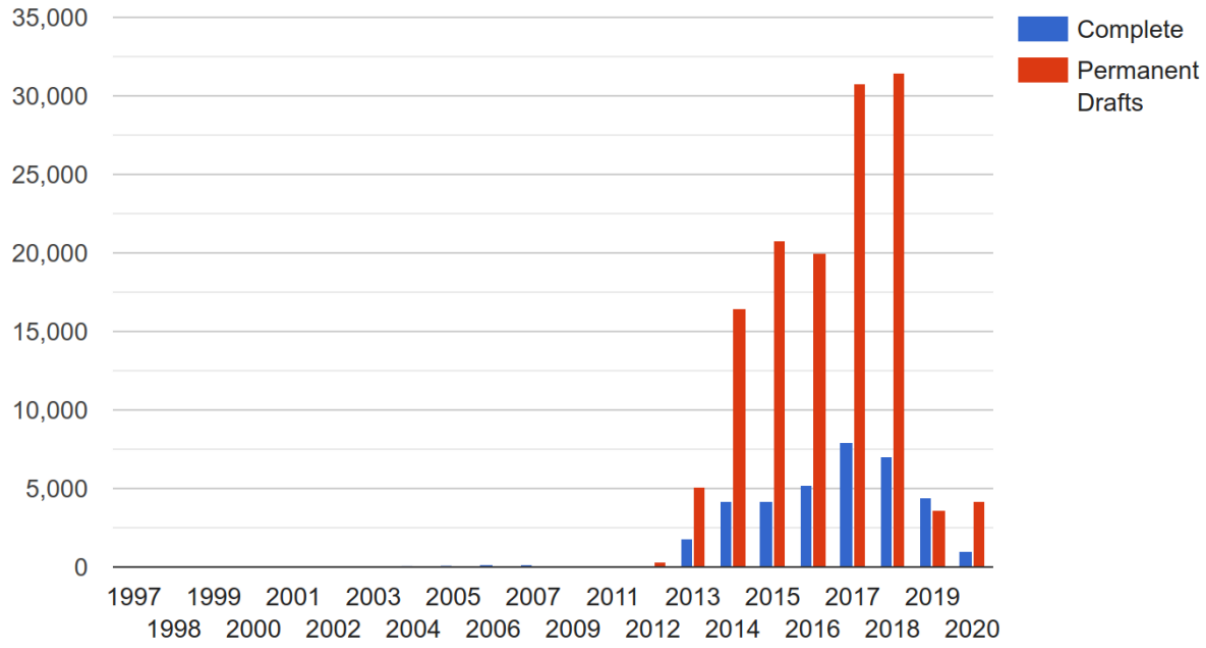
We have a database with as many sequences as we can. But how complete are those sequences?

We can check the taxonomic completeness (which species groups are more abundant in the database):

- We have a lot of bacterial genomes (175.000 genomes), because they are small
- We have less eukaryotic genomes (60.000 genomes)
- We have even less viral genomes (10.000 genomes and we estimate that there are over a million species of virus)
- We have even less archaeal genomes (1.000 genomes)

So, there is a big gap between the different taxons.

Here we have a plot that shows how many genomes are complete.
Permanent drafts means that nobody is going to spend more money in order to finish the genome. Maybe we have 100.000 scaffolds that need to be connected in order to construct the chromosomes.



This is going to change with the BioEarth genome project. Its aim is to sequence 1 million genomes. They imposed some rules:

- In order to be archived, it needs to be a complete genome at chromosome level.

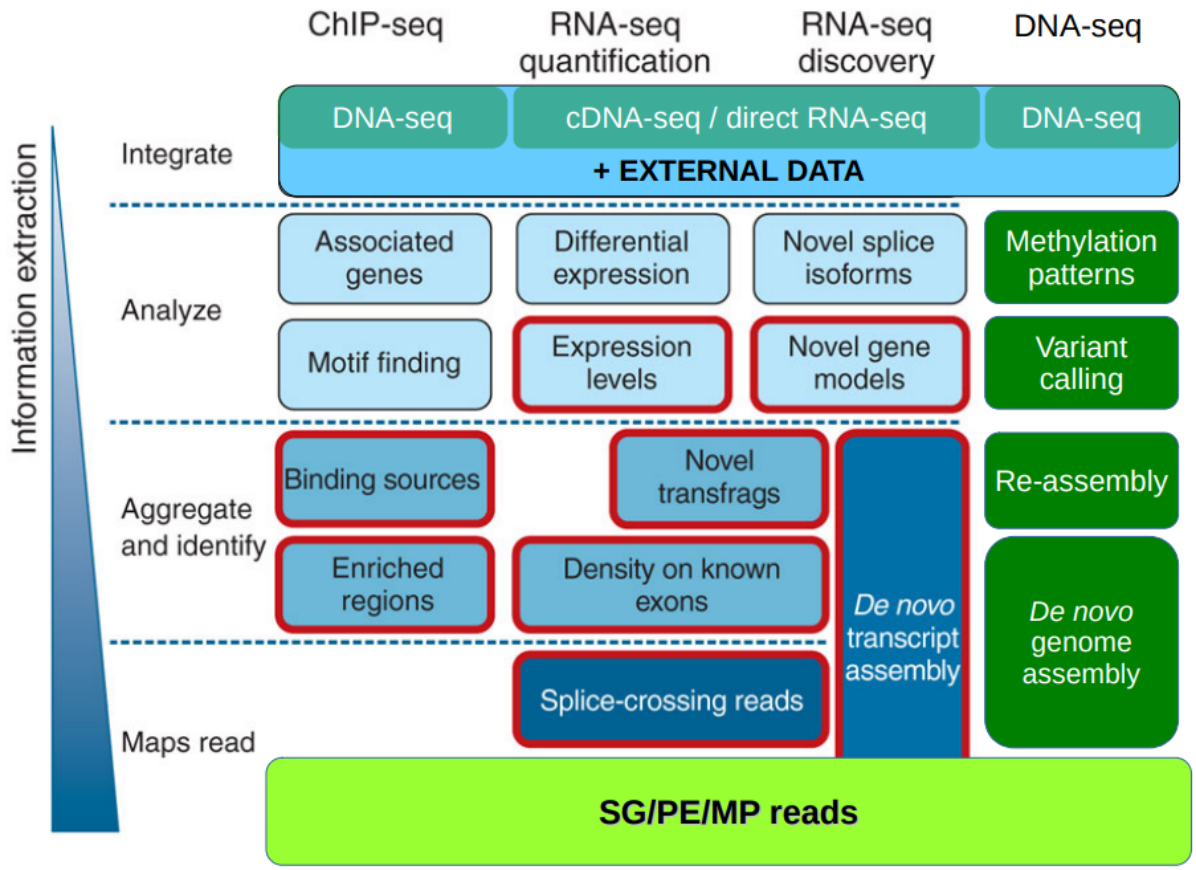
But still, most of the genomes we have are incomplete now-a-days.

Sequencing Overview

All sequencing technologies are equivalent. If we have DNA, we can obtain a series of reads (fragments we obtain from the sequencing device).

If we work with the transcriptome, we have mRNA. We need to transform it into cDNA and then sequence it.

All sequencing devices use DNA because it's cheaper.



SG: Single end reads; PE: Pair end reads; MP: Mixed end reads

If we are focusing a genome, we can use the reads to assemble de novo (I don't have a reference genome, so I try to connect the reads in order to construct the genome) or, if we have a reference genome, we can use them to reassemble (close gaps or fix mistakes of an annotated genome).

So, we can detect SNPs when comparing a sequence against the reference genome. Maybe these SNPs or other mutations don't affect a coding region but a regulatory region. This would also be useful because we can detect if a gene is more or less expressed and if this leads to a disease.

Maybe a disease is caused by a combination of many of those mutations that, at the beginning, looked independent.

The reads obtained from an RNA-seq experiment can be used to assemble a transcriptome to detect variants of splicing, genes that are expressed under a certain condition and time… The transcriptome is dynamic.

If we have different samples, we can compare the different levels of the samples and do a differential gene expression analysis under a certain condition.

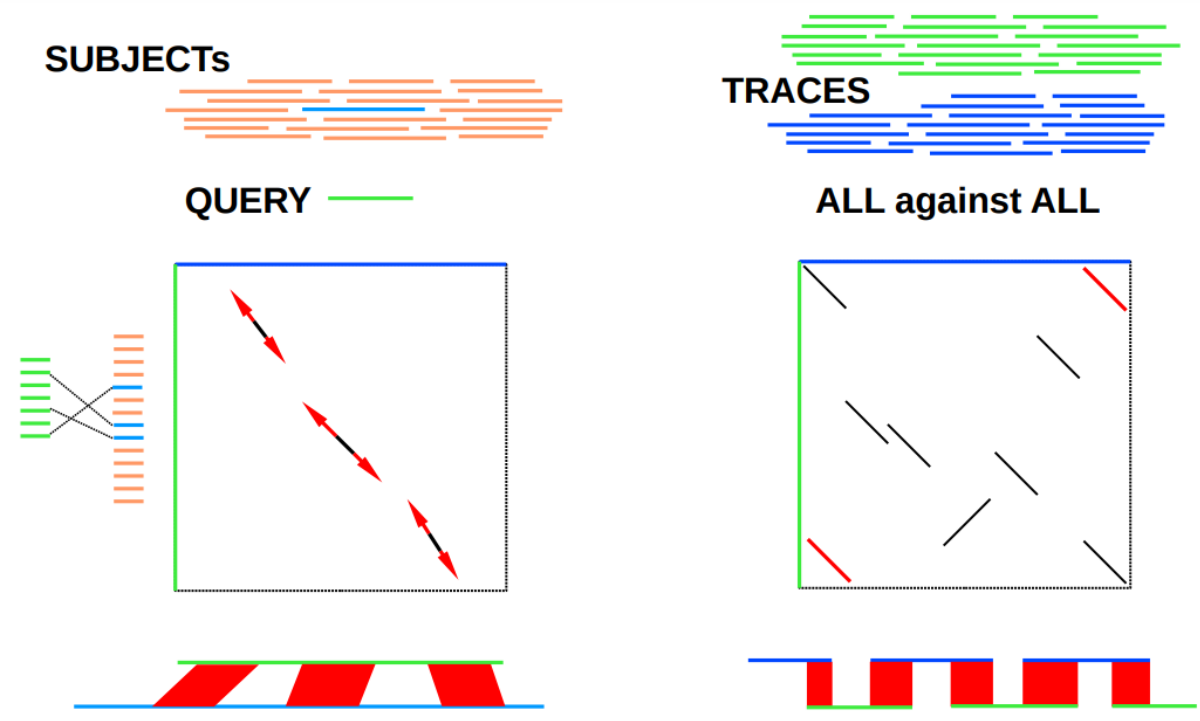
When using CHIP-SEQ we want to understand gene regulation, which is mediated by the transcription factors. So, we want to detect which are the genome regions that are recognized by the transcription factors (transcription factor binding sites).
How can we guess the sequence of these regions? We chemically bind the protein to the DNA and we degrade the DNA. The DNA that is bound to the protein is going to be protected and therefore we extract that DNA and just sequence the remaining DNA.

Now, we map that DNA to the reference genome and we will know all the places in the genome that are recognized by that TF.
If we know when this TF is activated, we can look for the genes that are downstream and know when this gene is regulated.

Assembly vs Homology

Homology search: We have a query sequence that we want to compare against a database of sequences. Maybe the query sequence is similar to the blue one.

When we are assembling, we have to compare all sequences against all the database sequences. This will take a lot of time and we need to make some shortcuts (k-mer analysis, indexing the sequences…).



Indexing a sequence means getting a fragment of the sequence:

- If we have one sequence (homology) we use only that
- If we have many sequences (assembly), we get an index of all the sequences. We compare the indexes of the sequence with the indexes of the database.

Shortcut: We are not going to align all the sequences, but I have those indexes that are more or less equivalent. So, I can just guess, from the sequences that are in common, which are the seeds from which I can grow the alignment.

This is how BLAST works. BLAST looks for these indexes that are going to be used as seeds and then it will expand them.

When doing the ALL AGAINST ALL, we are interested in the alignments that are at the 3' 5' ends of the sequences. So, I am only interested in those that allow me to connect one sequence after the other to reconstruct the chromosome.

RNA-Seq overview

We are sequencing the mRNA.
When mapping them into the genome, we will only find exons. Since introns are not transcribed.

So, we have a continuous sequence that can be mapped into the reference genome and will only cover the exonic regions.
There are 2 types of reads:

- Complete aligned reads that support the exons
- Spliced reads or junction reads that, one part of the read aligns to one exon and the other part of the read aligns to the next exon.

These spliced reads give support for the introns, we can detect them. Thus, we can define how many isoforms are encoded in a gene (maybe one has 2 exons and the other has 3 exons).

We can also check the coverage (how many reads align for that exon) and know the expression level of each isoform. Thus, we can know which isoform is more abundant.

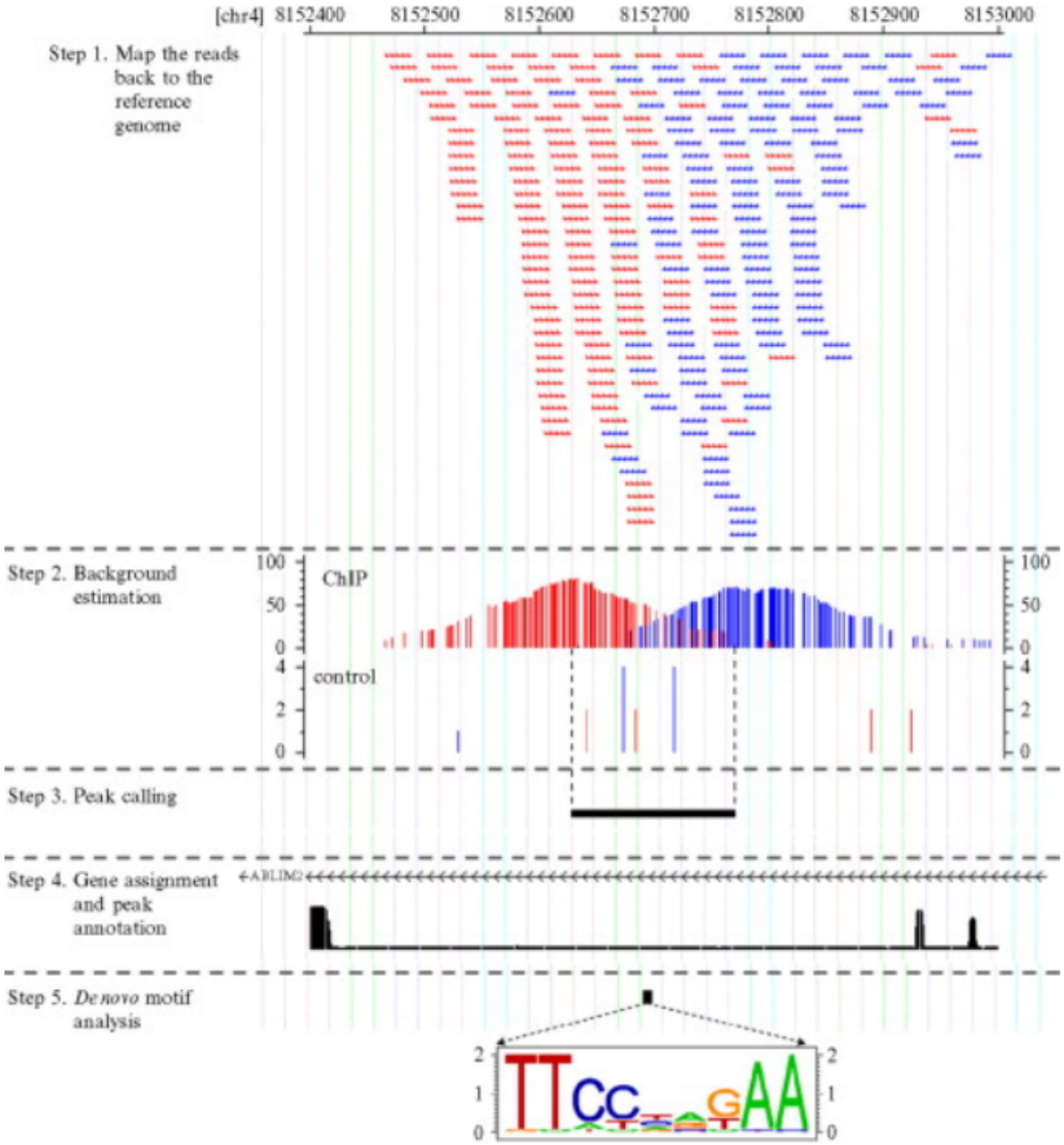
So, we can deduce the structure and expression level of an isoform.

From Reads to Peaks to Signals

What happens with the data we obtain from CHIP-Seq? Remember that we have the fragment of the DNA that was protected because it was bound to the TF. When we map those reads, we expect them to map to the regions that are recognized by that TF.

So, we get this type of pics in specific locations of the genome. In the middle of these regions of the pic we will find the sequence that is recognized by the TF.

Note that we have the forward and reverse.



Chromatin Conformation Overview

Estimating how chromatin is folded is also based on sequencing.
DNA is not randomly folded in the nucleus (it has some structure).

There are some proteins that interact with the DNA and define how the chromatin is folded. For example, histones.

We can use the same approach as with the CHIP-Seq and bind the DNA to the histons or other compacting proteins. Then we degrade the DNA and purify the DNA that was protected and thus not degraded.

We can also find distant relationships. Maybe the compacting proteins put close together two regions of the DNA that are distant (even different chromosomes). So, when obtaining the purified DNA and mapping into the genome we will observe that these regions are really distant. This could be because one region is an enhancer, for example.

There are different types of analysis that, at the end, determine segments of the genome that are together. We can map them into the genome and find those long range relationships.

2nd Theoretical Class

Sequence Annotation

Having a setup with the capability of sequencing samples from any organisms allows us to generate a massive amount of sequences that we can store in the databases.

But those sequences by themselves are useless. So, we have to process them in order to extract information.

When we are trying to understand a process, we start with raw data and then we generate models to have a higher level of understanding of what we are analyzing.

The real value of the omics (genomics, transcriptomics…) is not the capability of having the sequences but the capability of annotating those sequences.

What is an annotation?

We start with an anonymous sequence that we will call a query sequence.
The first thing we do is transform the query sequence into a coordinate system. So, for each nucleotide of the sequence we have a coordinate.

If we assemble a genome, we get a different sequence and, hence, the coordinate system changes with the sequence.

We know that probably this query sequence encodes for a function, which is characterized or delimited by some specific signals. For example, in eukaryotes, exons are surrounded by signals that allow its detection by the splicing machinery (so they can produce the chemical reaction to remove the introns…).

How can we detect those signals?

We always start with signal evidence. So, we have some experimental background, genes that have already been sequenced or analyzed… and as we move forward, we have more and more sequences in the databases.

We will focus on the sequences that have already been annotated.
Imagine that someone has sequenced a kinase and, hence they know that this sequence encodes for a kinase protein. Meaning that someone has experimentally determined that this sequence when it's translated into a protein produces that reaction.
So, we have some experimental validations that allow us to characterize the chemical properties of our query sequence by comparing it with the annotated sequences.

We also have genes from which we already know the gene structure, for example.

As we can see, we have databases that contain annotated sequences (sequence evidences) that contain signals that have been annotated (like the AG signal that corresponds to a splice site) and sequences that may have functional properties like protein kinases, channels, transmembrane proteins…

We can use 2 different approaches to annotate our query sequence using already annotated sequences:

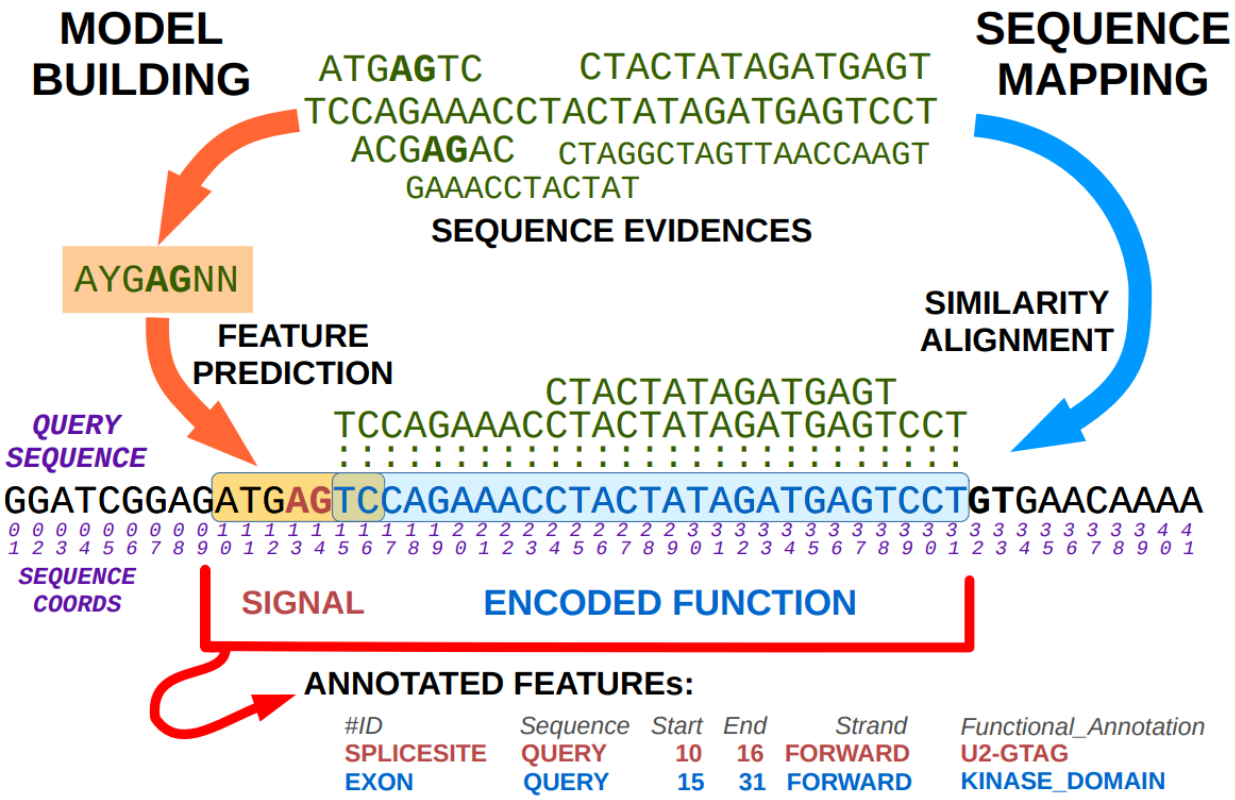
- **Easy approach:** Make a BLAST and map our query sequence against the database. We will obtain some alignments.
  Note that we do not define the process as “Sequence Alignment” but as “Sequence Mapping” because sequence mapping is more general.
  When we are performing an alignment we assume that we are comparing 2 sequences or more and applying a model that includes substitutions, indels.. like the Smith Waterman or Needleman Wunsch models in which we are including those biological concepts (so, we can model gaps).

  Sequence mapping is more general. There are other approaches (k-mers, for example) that at the end produce an alignment much faster.

  If the aligned sequences are proteins or functional elements, only in those places in which we have an alignment, we can guess a function encoded.
  So, once we have done the mapping we are transferring the annotation from the known sequence to the query sequence.

- Hard approach: Build a model. **Can we extract information from the known sequences in order to predict if that function or signal is going to appear into the query sequence?** In this case, we need 2 steps:
  - Process the known sequences. For example, we can make a model to predict the splicing sites. So, from known sequences I can make a model that somehow summarizes all the information we have for that signal (looks for a pattern, “AYG**AG**NN” in this case).
  - Once I have the model, I have to scan the query sequence with the model. Meaning that I take the query sequence and predict where the putative signals are encoded by taking the model and scanning the sequence nucleotide by nucleotide.
    If we have a probabilistic model, we will have a score. If the fragment of the query sequence is really similar to the pattern, it will have a higher score.
    At the end, I will have a score for every position and I can define a threshold and filter the fragments.

    The more sequences we use to build the model the better, since it will be more accurate when predicting the signals.



We need 4 elements to define an annotation:

- **Sequence ID**, because the coordinate system is relative to the first nucleotide (so it depends on each sequence). Thus, we will always have to define the sequence ID (maybe the sequence has been updated, for example).
- **Start** of the sequence
- **End** of the sequence
- If we are annotating a genomic sequence, a priori we do not know if it's the **forward or reverse strand**. So, we need another field to define the direction.

So, these are the minimal elements of information that define a segment in a gene.
We could also add another field that specifies a little description ID. For example, if it's a splice site, exon, intron, regulatory element…
Also we could add a functional annotation.

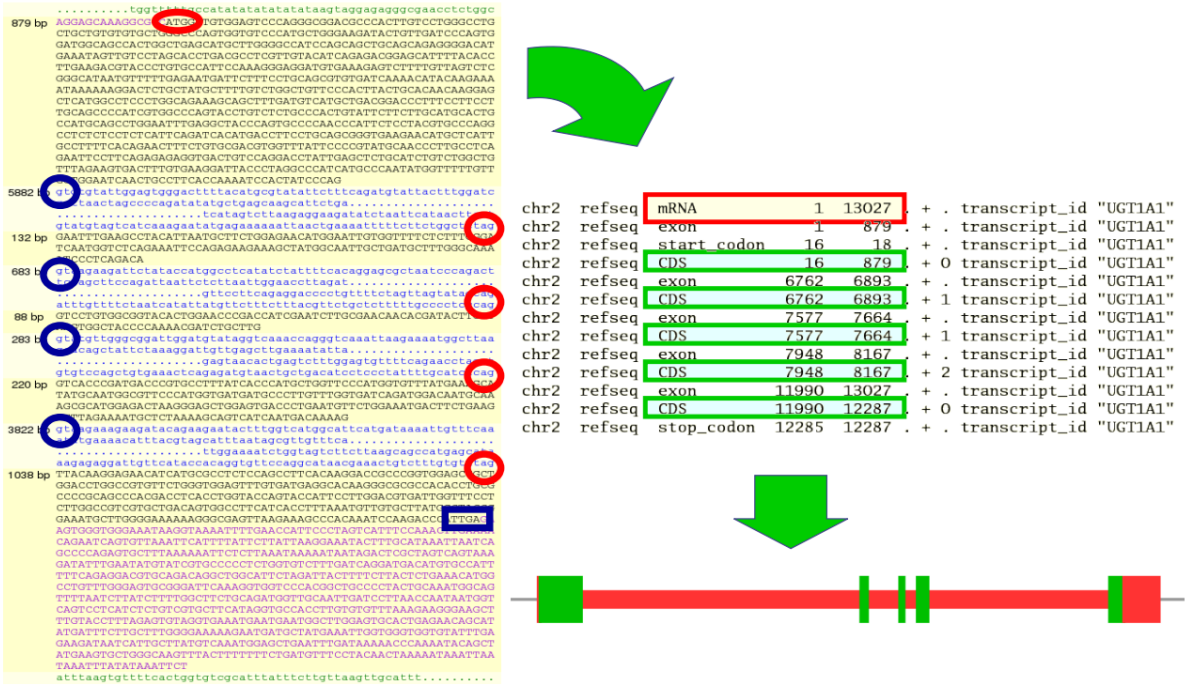
If we are annotating proteins, we do not need to add the strand field (because the peptidic bond already puts a directionality from the amino terminal to the carboxyl terminal).
For transcriptomic data we need the strand field because we will have to transform the mRNA into cDNA and we need to know which strand we are analyzing.

Drawbacks of building a model:

- We will need a model for every signal. For example, one model for the donor splice site, one for the acceptor splice site, one for the start of the exon…
- We know that species evolve and sequences change. If we pick close enough species, we can use models for both species. For example, we build a model for rats and we also use it for mice or even rabbits.
  So, we can use a general model and use it with several species. Obviously, if we use the model in a species that is not that similar, we will obtain more false positives/negatives and, hence, we will obtain a poor annotation.

The next step of the annotation of a genome is to transform the coordinates to a graphical metaphor.
Thanks to this we will be able to see which is the longest/shortest intron/exon, which is the first one, the last one… Otherwise it's very difficult to do by hand.

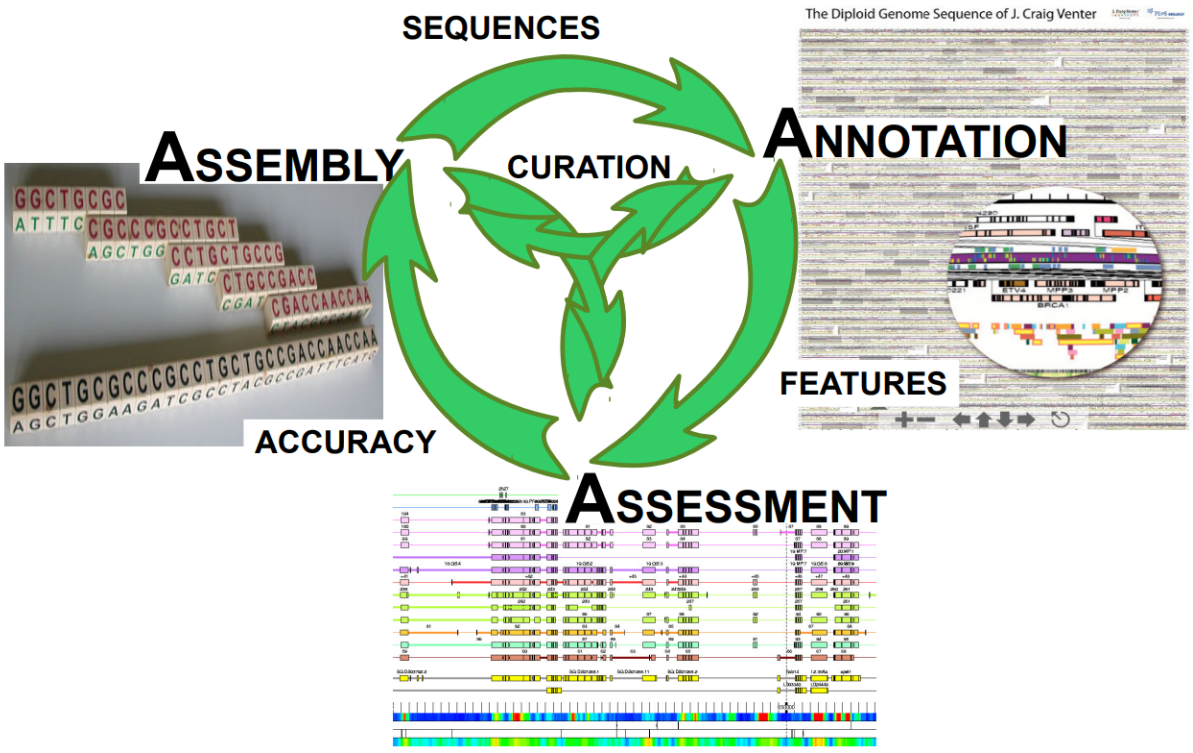
Recap: Raw data (sequence) is transformed into a series of annotations using a model and then we transform these annotations into a graphic visualization.



This is not a single run analysis.
We can have new reads from sequencing experiments and, hence, we need to assemble them and we will obtain a new version of the genome (the sequences have changed a little bit).

Every time that we have a new assembly, we need to annotate the new sequences.

Once we have done an annotation over a genome, there is a process of validation to check if the annotations are correct. The models are not 100% accurate.



The assessment of the accuracy gives us an idea of the error rate for the prediction but does not tell us which are the predictions that are wrong.
We can use experimental data to validate those predictions.

The new predictions can be introduced in the database to build an updated model…

Assembly vs annotation versions

Genome browsers are based on a genome version:

- Human genome version GRCh37.p10

P10 means patch 10. It's the subversion 10 of the human genome 37.
Meaning that they have resequenced and closed some gaps, corrected mistakes… that have been patched in the reference sequence. But there were not enough changes to release a new version of the genome.

On top of that, we have the version of the database for the annotations.
For the same genome version you can change the parameters of a program, so you can recalculate the analysis.
Thus, you can have different versions for the same genome version.

So, the versions of the annotations can differ from the versions of the sequence.

Most of the databases or browsers have information about the pipeline they used to make the annotations.

If we annotate a gene, a priori we don't know if it's a coding, non-coding gene or a pseudogene. So, depending on the model we have used we will obtain a result or another. Because there are specific biases in coding genes that we can use to better predict the coding genes.

Annotation layers

Remember that annotation is as easy as defining segments:

- Start and end in the genome.

We start with a version of the genome and the first thing we do is find some statistical properties of that genome, like the GC content (NOT where are the exons, introns…). So, we look for the regions that have a higher or lower GC content.

We do this because there is a correlation between coding regions and high GC content.
So, having this characterization allows us also to adjust the models.
If we know that the GC content of the genome varies with the GC content that we use in the model, we can adjust some parameters and solve this problem. Otherwise we will get false positives/negatives.

Imagine that we have a genome with a GC content of 70% and we are using a model that uses a GC content of 40%. The model will tell us that there are a lot of coding regions in the genome because it finds a high GC content. But this will be wrong.

Once we have characterized some statistical properties of the genome, we annotate the repetitive sequences:

- Microsatellites
- Simple repeats

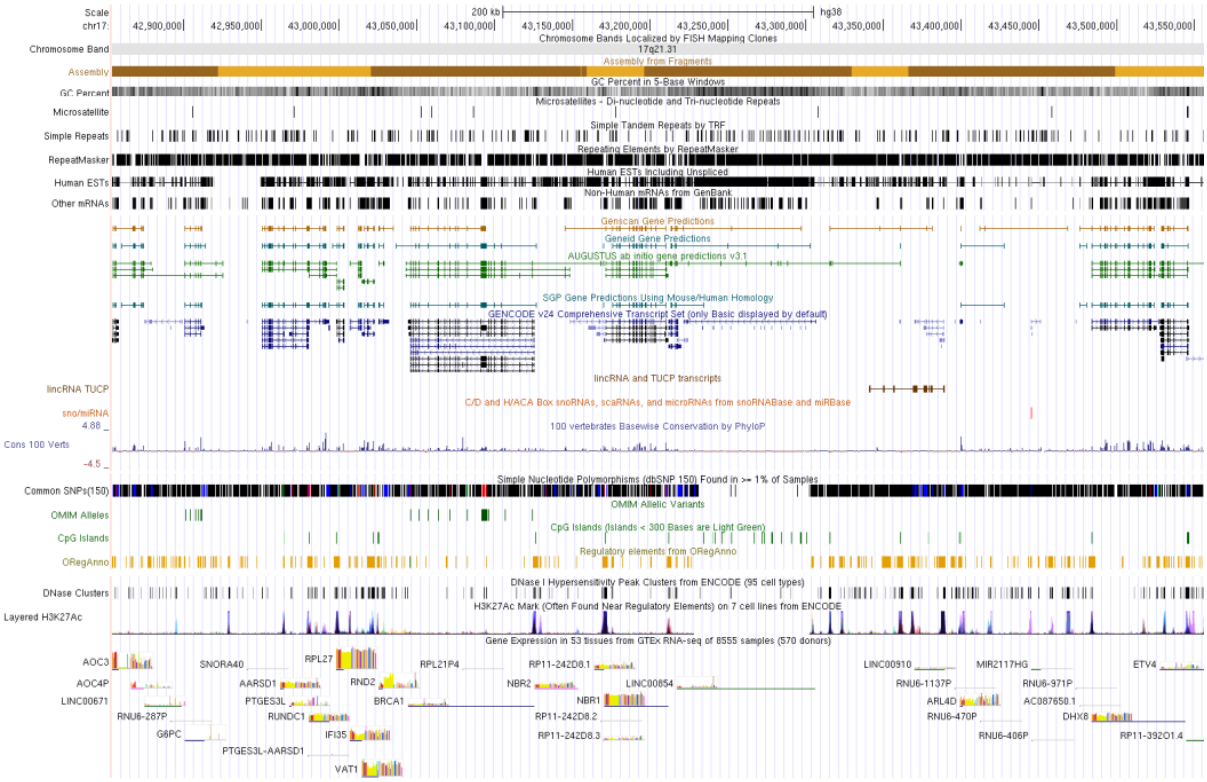
This allows us to mask regions having a repetitive sequence. Because we know that repetitive elements are "junk" DNA, because they have no functions.
So, if we have nucleotides encoded in the segments where we have repeats we just replace the nucleotides for that repetitive region into NNNN. So, the programs that are going to look for genes don't get wrong predictions.
We will reduce the number of wrong positives because if we find something similar to a splice site in a repetitive region, we mask the repetitive region and we will not have a false splice site predicted in that region.

Also it's going to speed up the annotation because we have many repetitive regions that we are going to pass. 50% of our genome are repetitive elements.

Once we have the genome masked, we can start running programs that are specialized on the annotation.

When comparing genomes of different species, we can see that there are some regions that are conserved… This is due to the fact that there is a really important gene codified in that region that can not be mutated because it will generate a biological disadvantage to the individual.

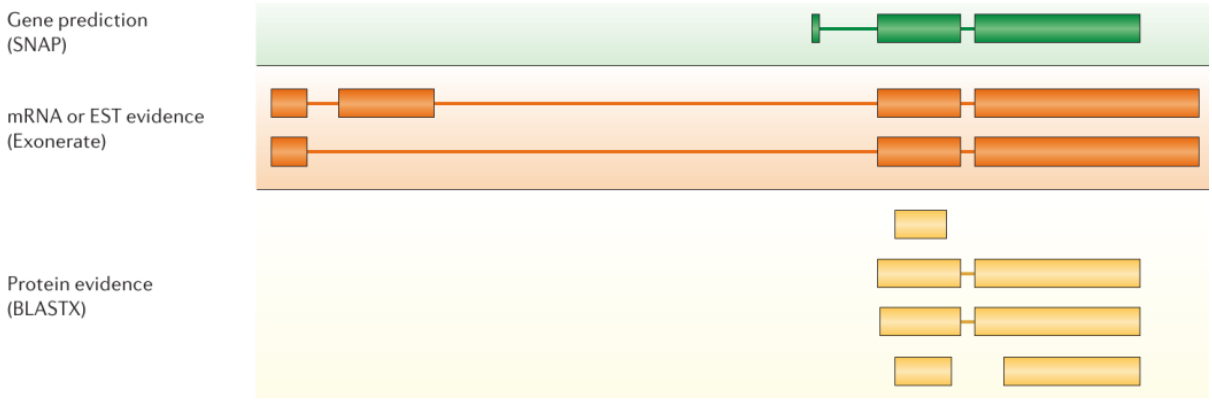
Min 39 Comput 2 explica aquesta imatge



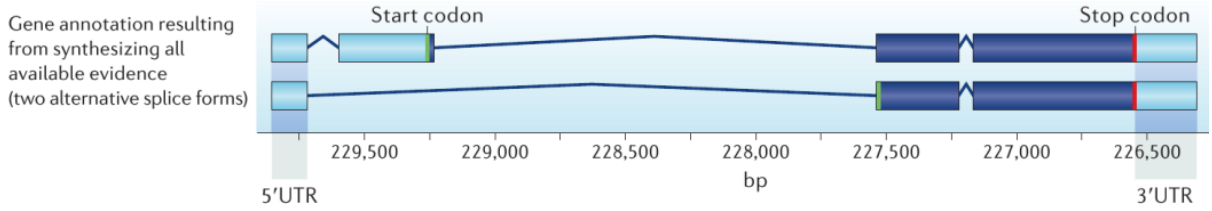
Gene-finding vs Gene annotation

Gene finding is using tools in an automatic way (genomewide). So, we take a genome and we annotate using BLAST, SNAP, Exonerate:

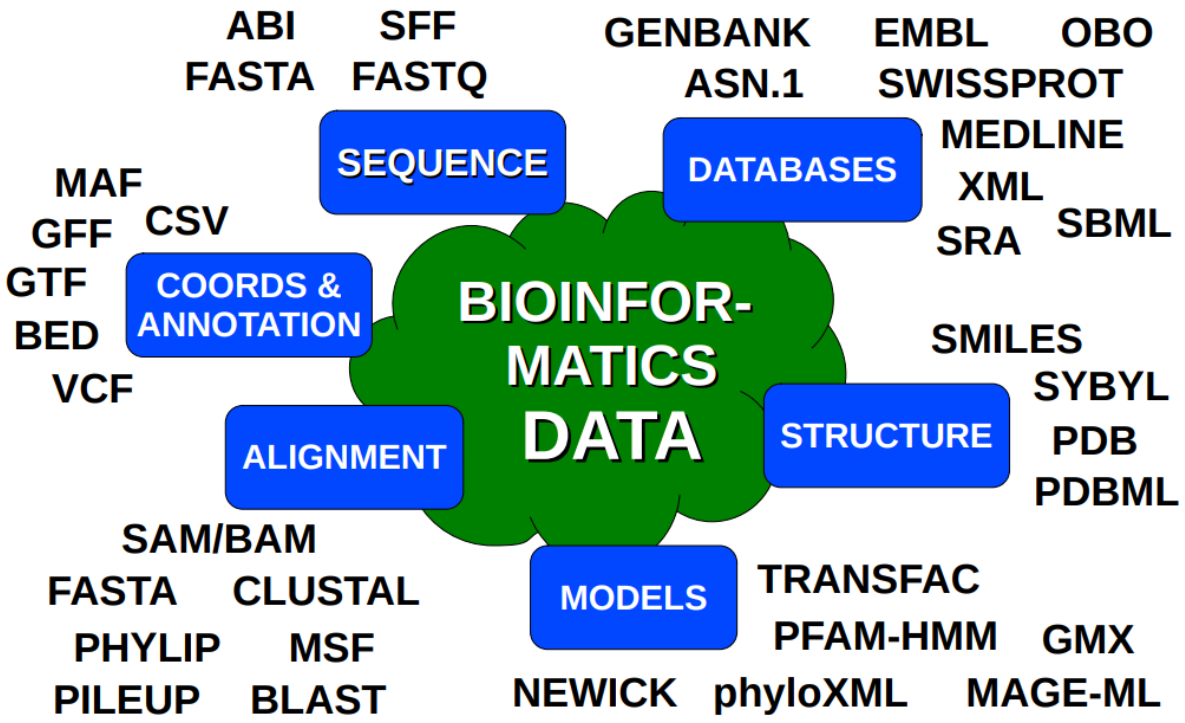
- BLAST looks for similarity regions. It does not know anything about splice signals… If there is conservation, we will have a hit. But it does not tell us something about the structure of the gene.
- Exonerate is a specialized alignment tool. It does the same as BLAST, but it takes into account that the alignment can be broken at the introns. So, it has a model for the splice sites (but it's still an alignment tool).
  It's aligning but when it finds a splice site, it stops aligning and looks for the next splice site and then continues aligning.
- SNAP is a gene predictor tool that can take into account homology, alignments, signals… but also it has a gene model. So, it does not only find the exons but also how they are connected to define a gene.



Gene annotation: A human curator can take all the evidence obtained from the different programs, look for coincidences and refine the final annotation.



Data formats



As we can see, we have many formats that fit for a type of data, models…

It is important to distinguish between tabular formats and multiline sequence records.
In this case we have the 2 formats with the same information:

- Sequence Id
- Nucleotide sequence
- Extra fields

Why do we need two different formats to store the data?
It depends on the application.
If we have data in tabular format, the records are going to be easy to filter out. I can simply write a simple grep, python/perl script… that processes line by line and filters out the sequences that match a pattern, for example.

Multiline sequence records allow us to skip some limitations of the program.
What happens with sequences that are large? For instance, if we have a genome annotation we will have the annotation of the different chromosomes.
For chr I, we will have 250 billion nucleotides and, hence 250 billion characters and a new line at the end in a tabular format. Many programs have a limit of how many characters can read per line.
So, FASTA is a good format to store very long sequences because we separate the sequence in fragments of a fixed width that end with a new line.
Thus, the programs with a limited buffer can read the whole sequence.

Short sequences are more suitable for tabular formats and long sequences are more suitable for multiline sequence records (like FASTA).

What happens if I am looking for a pattern in a FASTA? That our pattern can be splitted.
Thus, we need to reconstruct the data:

- Read the fragments
- Reconstruct the sequence
- Apply the model that looks for the pattern

We have direct access for the tabular format and 2 step access for the multiline sequence records.

At the end, knowing the specifications of the formats allow us to process the information efficiently and to chose which is the most appropriate format for our analysis:

- If we are analyzing splice sites, we are going to work with hundreds of sequences of 20 to 100 nucleotides and, hence, tabular format is very efficient to store the sequences of the splice sites.

Sequence Ontology (SO)

What is important is to define the features of a sequence.
We are annotating segments:

- We have a starting coordinate based on a reference coordinate system
- We have a strand if we are coworking with genomic data
- We also have the feature ID: What we are annotating.
  So, we have series of elements that we annotate over a genome, for instance:
  - Gene
  - Transcript that is encoded within a gene
  - mRNA, which is the processed transcript
  - Exons
  - Etc

So, there are different features that we can annotate on a genome, but also there is a hierarchy that depends on the level of structure.

- Genes are on top of transcripts, since a gene can have multiple transcripts
- A transcript is made of multiple exons

So, we have levels of annotation and we have different features to annotate.

What is the best way to work with different sets of annotations? That everybody agrees on the meaning of each word (no ambiguity).
This is why we use Ontology → Formuled dictionaries or vocabulary.

So, we have a term (annotation feature in this case) that is linked to a unique code that facilitates computational searches, because searching by strings is slower than searching by numbers.
If we can assign a unique number to this term, we can perform automatic searches by the code. So, for example, we can search in the database which are the features that contain an exon instead of looking for the string “exon”.

Important:

- **Gene Ontology (GO):** Set of terms and relationships used to describe the function of a gene.
- **Sequence Ontology (SO):** Set of terms and relationships used to describe the features and attributes of a biological sequence.

Strandness

There are other issues that can appear when we are annotating features on a genomic sequence. The worst thing that can happen is that you are annotating a genomic sequence and you do not take into account the strand.

If a gene is in the forward strand, the polymerase will use the reverse strand as a template and thus we will obtain a transcript that is the same as the coding strand.

But DNA can encode features in both strands. If we do not take into account the strand, we may have problems.

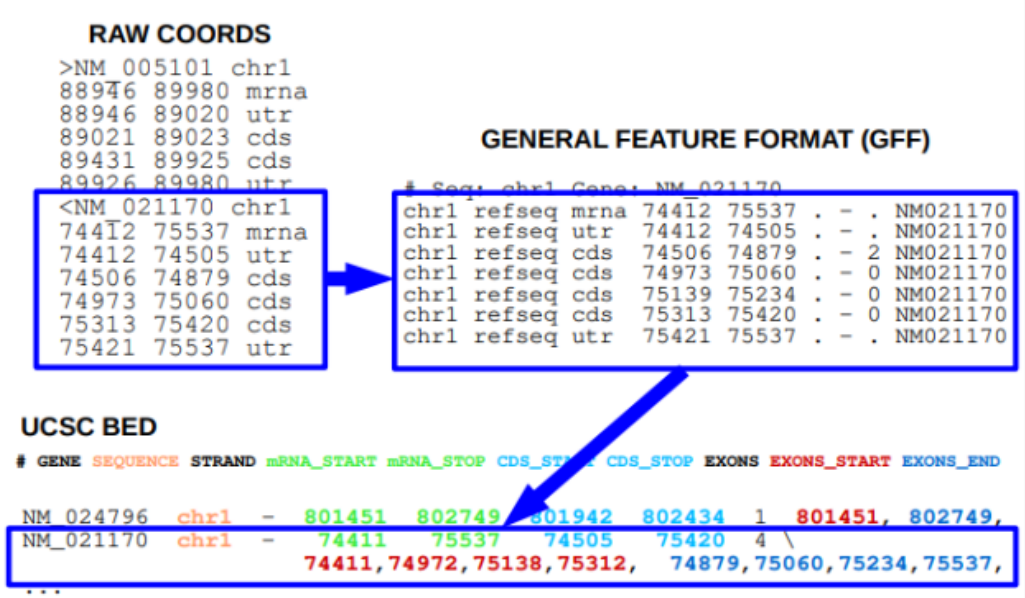
All the sequences that we can find in the databases are in the 5’ - 3’ direction.
Thus, to store the reverse strand we must make the complementary of the forward and then reverse it.

Annotation formats

There are many sequence (see before) and annotation formats:

- Multialign record format, similar to a FASTA.
- Tabular format like the General Feature Format (GFF)
- UCSC BED format, that focuses on the gene. So, a single record or line contains only information for one gene structure. Used for genome browsers, because they have thousands of genes.

These formats are equivalent, so we can easily make a script that changes from one format to another.



Strands & Annotation Coords

We have different ways to encode the coordinates in the different formats.
So, the formats need to define the start and end coordinates and the strand.

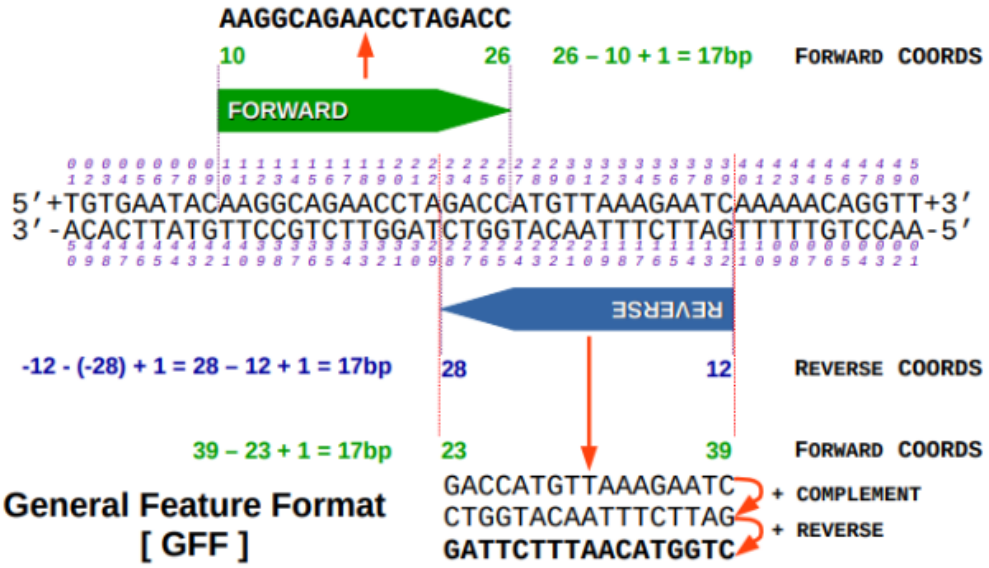
As mentioned before, it is important to have in mind that:

- Forward strand is the 5'-3'
- Reverse strand is the 5'-3' of the reverse complement of the forward strand.

This affects the coordinates. Since we annotate features using a set of coordinates based on the beginning of the sequence.
So, we can annotate the coordinates of the reverse strand in 2 different ways:

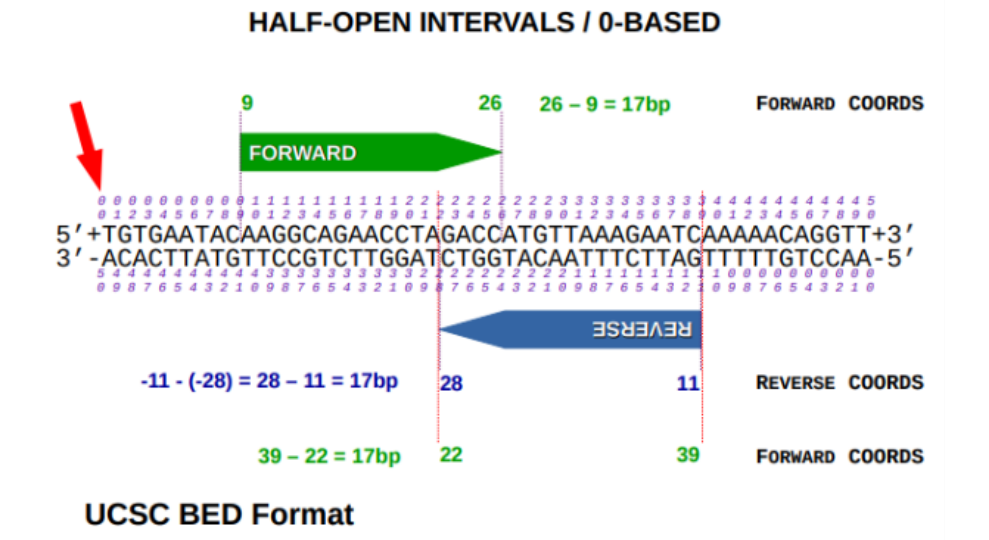
- Reverse coordinates
- Forward coordinates

If our program is always using forward coordinates, we need to be aware of the +/- strand in order to know the proper orientation of the features we annotated in the sequence.



If we know the specifications of the annotation format, we will be aware of this. For instance, GFF format is a format that encodes the coordinates always in forward. Thus, we will have a column that specifies the strand.

There is another constraint or property for the GFF format: It is a fully closed interval, meaning that the segments start in position 1. So, we are including the starting and ending coordinate of the feature.
If we want to compute the length, we need to do the difference (26-10) and add +1 (because the number 10, in our case, is also part of the feature).
We need to add a correction!



Here we have the UCSC BED format, which has half-open intervals. Now instead of starting from the first nucleotide, we start from 0 and the coordinates are not on any single nucleotide but between nucleotides. Thus, the numbering coordinates do not correspond to positions of the nucleotides.
This simplifies our calculations because we do not need to add +1 to calculate lengths.

Complex Records: GenBank

When we have to combine different kinds of information we require complex record structures like the Genbank format.

We have labels that describe the features.

```
LOCUS       NM_021170 893 bp mRNA linear PRI 24-SEP-2005
DEFINITION  Homo sapiens hairy and enhancer
            of split 4 (Drosophila)(HES4), mRNA.
ACCESSION   NM_021170
VERSION     NM_021170.2 GI:20127596
...
SOURCE      Homo sapiens (human)
FEATURES    Location/Qualifiers
  source    1..893
            /organism="Homo sapiens"
            /mol_type="mRNA"
            /db_xref="taxon:9606"
            /chromosome="1"
            /map="1p36.33"
  gene      1..893
            /gene="HES4"
            /db_xref="GeneID:57801"

  ...
  CDS       118..783
            /gene="HES4"
            /note="bHLH factor Hes4"
            /codon_start=1
            /product="hairy and enhancer of split 4"
            /protein_id="NP_066993.1"
            /db_xref="GI:10863967"
            ...
            /translation="MAADTPGKPSASPMAGAPASASRTPDKP
                          ...
                        LLPGLTRALPAAPRAGPQGPGGPWRPWLR"

  ...
ORIGIN
        1 gggaaagaat gcggagccgg gttcacacac cccgcggcgg cgaggcctta aatagggaaa
       61 cggcctgagg cgcgcgcggg cctggagccg ggatccgccc taggggctcg gatcgccgcg

          ...

      901 ccgttctagg gccgtggcct ttgccgagac tgtagcagag aaaacgtatt tattattcca
      961 ga
//
```

Nowadays we do not use this format because it is very slow to process.

Sequence statistics

Basic Genome Stats

One of the first things to do with the sequences is just to take some analysis about the GC content, repetitive sequences…

The genome sequences can vary a lot depending on the species. They differ in:

- Length
- Number of genes
- GC content

Viruses are a parasitic organism and, hence, they do not require a lot of machinery since they use the one from the host. So, they require a few genes (low complexity).

Humans have a bigger genome. **Does this mean that having a bigger genome we are more complex?** Having a larger genome implies that you can encode for more genes, so maybe there is a correlation between having more nucleotides and the complexity.
The answer is NO.
There is a relation between the complexity and the regulation of the organism (network of interaction between the genes).

C-Value Paradox

There is a c-value paradox that says the following: There is no correlation between the genome length and the complexity.

GC Content affects Tm

There are other features, like GC content, that we can estimate from the genome:

- GC content of the whole genome
- GC content for some specific regions. We can see that there is a big variation between different regions.

Why is it important to take GC content into account? Because the GC content implies a specific amount of pairings that are different from AT. So, GC defines 3 hydrogen bonds and thus it is more stable.

When we are hybridizing molecules, for example in a PCR or sequencing experiments, we have a fixed temperature to separate the DNA. Thus, all this reactions depend on the GC content of the sequence:

- If it has a high GC content it will need a higher temperature

So, variation on the GC content may have an impact in the hybridization, which is measured by the melting temperature (Tm). Which is the temperature at which half of the molecules are denaturalized or hybridized.

Hence, if we have a mixture of regions that have high and low GC content, those regions may be amplified more efficiently by PCR or not. **So, it affects any experimental procedure based on base pairing.**

%GC on Coding Sequences

Coding regions have different GC content than non-coding regions. Often, coding regions have a higher GC content than intergenic regions and introns.

Especially in species that have low GC content on average. Because in some cases all GC content will be present in the coding regions.

Chromosome 19 has a high density of genes per megabase and this correlates with the GC content (it also has the highest GC content). But the GC content is not uniform, for example, centromeric regions have poor GC content (because it's poor in genes).

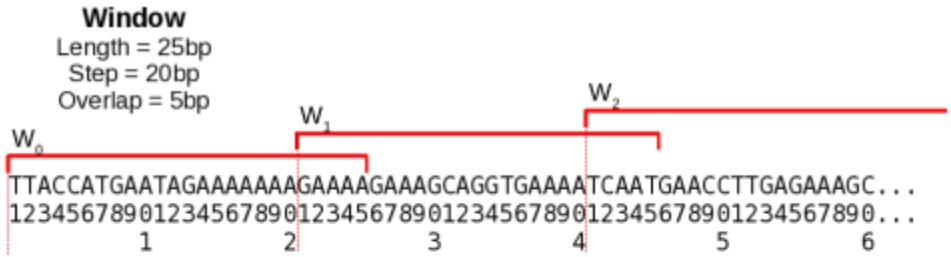
Running Windows

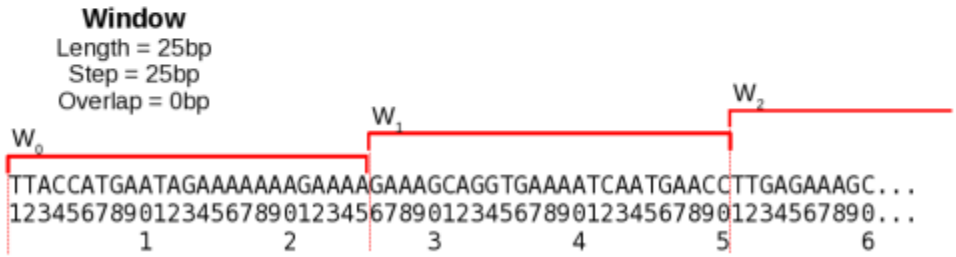
How can we analyze the average GC content along the genome?

We can just compute the proportion of G and C by counting the number of times a G or C appears in the whole genome and divide the count by the length of the genome. This is like calculating a window of the size of the whole genome.

If we want to look for local variations, we have to apply some technique to analyze that. The easiest technique is the running window:
We take strings of a given size (window length) and run this “string”, taking substrings of the genome, and computing the GC content for each of the windows.

Depending on the size of the window we will have a more detailed or a more average value of the GC content.



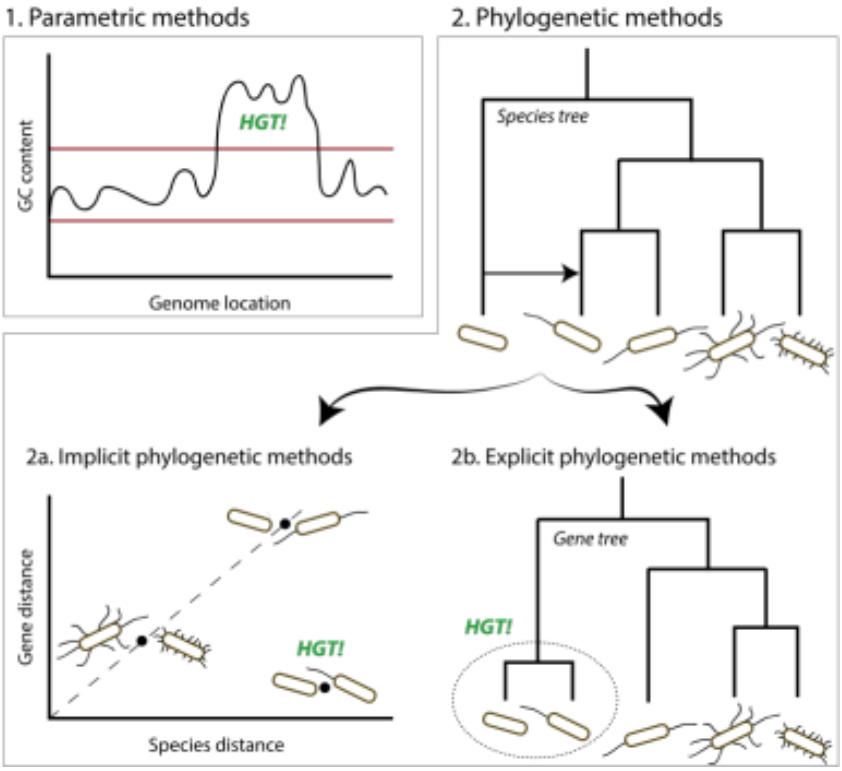


3rd Theoretical Class

%GC to detect HGT Events

By calculating the GC content along the genome, we can detect a rapid change in the GC content due to a horizontal gene transfer.

When comparing the species and gene tree, we can see some incongruences that are due to this HGT.



Strings of Symbols

We can easily process sequences in our computer because computers are quite good at analyzing sequences of 0 and 1.
So, we can translate any sequence of 0 and 1 to a series of symbols. For example:

- A is 0001
- C is 0011
- G is 0111
- T is 1111

At the end, we will have a series of symbols defining those strings.
We have to define an alphabet to generate strings of any size that can be composed of any symbol of the alphabet.

$$\Sigma_{DNA.\,IUPAC} = \{ a, c, g, t, u, r, y, m, k, w, s, b, d, h, v, n \}$$

$$\Sigma_{AminoAcids} = \{ a, r, n, d, c, q, e, g, h, i, l, k, m, f, p, s, t, w, y, v, b, z, x \}$$

Unit 2. Sequence Analysis

It is important to understand how the sequences are generated (sequencing technologies). We will also look at how the quality of the sequencing is calculated and the issues that can appear.

We will also focus on properties of genomic sequences like, how can we use k-mers to index sequences and specific data structures that facilitate the indexing.

Moreover, we will talk about measures of complexity. With this information we will be able to detect repetitive sequences. This is one of the important steps prior to annotating a genome. Remember that we have different layers of annotations:

1. Calculating statistics
2. Look for repetitive elements that can affect further calculations

Getting sequences

We want to recover sequences from biological samples. So, this will be the pipeline:

- Go to the field and capture an organism
- Take the organism to the lab and process it to retrieve different molecules. We can do omics on any of the extracted molecules:
    - **Metabolites** (glucose) can be analyzed using metabolomics. We can see which experimental condition changes the flow of energy or nutrients in the cell, by just analyzing the composition of glucose in the sample.
    - **Proteins, DNA and RNA** have a specific sequence of nucleotides or amino acids that define the molecular properties of that molecule. Any technique that allows us to analyze molecules can help us to discover the sequence of nucleotides/aa on those polymers.
        - We can use the tunnel effect microscope that scans the atoms in a sample.
- We bring the molecules to sequencing facilities and here we can use different methods:
    - **Sanger (nt) / Edman (aa) sequencing methods:** Sanger uses radioactive dNTPs. Edman (was discovered first, so we first sequenced proteins) depends on the degradation of the protein and detecting which was the last aa in the sequence.
      Edman is not as escalable as Sanger.
    - **Mass spectrometry** is specially used to discover the protein sequences, but also DNA or RNA.
    - **Microarrays** are useful to detect gene expression by detecting its variation…
      Basically we have a set of probes, that are complementary to the target sequence, attached to a matrix. We can create probes that have different SNPs and thus detect at which probe the sample binds with more efficiency (used to sequence a small number of nucleotides).
      Their statistical analysis is very robust.
      But they can only detect the sequences that have been designed in the probes. Thus we can only explore known sequences.

Moreover, we have a limitation on the saturation. If the spot where you have the probe for a given sequence is already filed, no more target sequences will be able to bind. Thus, it is complicated to quantificate.

- **Next Generation Sequencing** methods (NGS) were implemented as a way to perform the same process more efficiently.
    - **Pyrosequencing (454 Roche):** Adding a nucleotide emits light (pyrophosphatase) that can be detected. There is an enzymatic reaction that amplifies the signal.
    - **By Ligation** (Solid) is very expensive.
    - **By Synthesis (Illumina HiSeq/Miseq)**: It is the cheapest
    - **Single Molecule (PacBio SMRT/Oxford NanoPore)**: Obtain the sequence of a single DNA or RNA or protein molecule from a sample.
      So, we will obtain very long reads but they have a lot of errors.
      This can be useful for the BioEarth genome project since it requires chromosome level assemblies and to do that, we need single molecule long reads in order to complete or finish the assemblies.
      We do not really care about the errors, because they can be corrected if there is a high read coverage.

MS-Sequencing

This technique follows this protocol:

1. Take a sample of proteins that we want to identify (most of the time we use MS to sequence proteins) and separate and purify them using chromatography, for example.
2. Recover the single protein
3. Use this protein in the MS.
    a. The protein is going to be fragmented (using an enzyme) and ionized. Thus, we will have different size fragments with different atom compositions and, hence, it will have a different charge.
    b. Detect the pics, which depend on the ratio mass and charge of the fragment.
    c. Guess which was the original molecule by reading the pics.
        i. At the end, we do not know, for a given peptide fragment, which is the pic set that is associated with that amino acid pattern.
        If we have a fragment that is made of A-V-A, it will have the same weight and charge as a fragment made of V-A-A.
        So, we can detect the aa composition but not the order.

        Thus, once we have the pics that define one of those peptides, we take a second round of MS (double MS analysis) in which we check the fragment composition of the fragments.

        From that, we take a database of known proteins and break the known proteins using the same enzyme that we used to fragment our sample. For every single fragment generated, we compute the ratio mass/charge and generate the pics.

Finally we compare the simulated pics with the one we obtained from the double MS analysis and then obtain the possible composition of the peptides in the sample and then we use that to reconstruct the protein.

As we can see, it is a very laborious task that needs a lot of protein.

DNA Sequencing Technologies

We will focus on DNA because RNA is more unstable and, hence, we retrotranscribe it into cDNA for sequencing. For this reason most of the sequencing technologies are for DNA even though we are doing transcriptome analyses.

We can divide them into 3 categories according to their throughput and quality of the throughput:

- **1st generation** includes Sanger. It is based on getting many copies of the same molecule of the sample. So, we are analyzing samples that contain only one molecule and we tag sequences (we have tagged sequences)
  From the sample sequence we create copies by using NTPs and tagged dNTPs that will stop the sequencing reaction.
  Since the marked nucleotides are added randomly, from the original sample we obtain different copies of different lengths with the last nucleotide marked.

  This gives us a strong signal, because we have many copies of every single molecule.

  We are going to identify the position of each nucleotide by length. So, we will have a capilar gel in which we will run the fragments and separate by their size.

  It has low coverage and long reads

- **2nd generation** includes pyrosequencing, Illumina… We are detecting an incorporation of a tagged nucleotide (every type of nucleotide has the same fluorochrome). The polymerase adds the complementary nucleotides to the sample and, in each incorporation, a fluorochrome is released producing an amount of light that can be detected.
  So, instead of having tagged sequences we have tagged nucleotides that are identified by synthesis.

  The problem is that it only works with short sequences. In order to have the proper intensity measures of the spots, this sample molecule can not be too long (300 nt).
  Problem of repetitive nucleotides → We can not detect how many of them because the intensity of light is not clear.

  There is a high coverage (obtain a large number of reads per run).

- **3rd generation** methods are based on direct identification (no markers or tags). The molecule goes through a pore that detects the change in the membrane potential when the nucleotide sequence passes.

  We obtain very long reads (2 million nucleotides in a single read) and medium coverage. The length of the read depends on how well the sample DNA is conserved (we need a good quality).

Sanger vs NGS (Next Generation techniques)

As we said before, we can sequence a larger volume using NGS.

- We are analyzing millions of spots at parallel

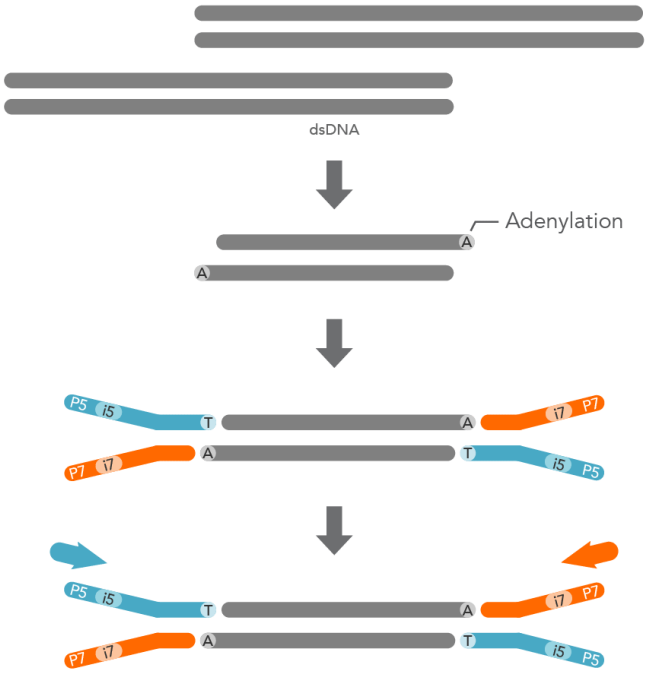
Also, the preparation of the sequencing libraries has improved.

For Sanger sequencing:
Just think about the Human Genome Project. They spent 10 years in order to develop the libraries of genomic fragments from a single individual in order to later on extract the fragments on those libraries for sanger sequencing.
If you break the genome into 1 million fragments, you have to keep those fragments cultured in bacterial colonies by transfecting them…
Then we will have a genotec (series of bacterial colonies that have a cloning vector that has an insert that corresponds to one of the fragments).
So, then we will have to deal with millions of clones and pick a single one, extract the fragment and then sanger sequence.

Moreover, we will need to be sure that the clone has the cloning vector and that the cloning vector has the correct insert. Then sequence one by one…
As we can see, it is very laborious, it took 10 years.

For NGS:
In order to get enough sequences to parallelize, we need to improve the yield of DNA molecules we can provide as sequencing libraries.
The biggest improvement in sequencing technologies is the usage of sequencing adapters (DNA molecules) that facilitate the linking of the DNA fragment to the sequencing slide and also that facilitates the initialization of the sequencing experiment.

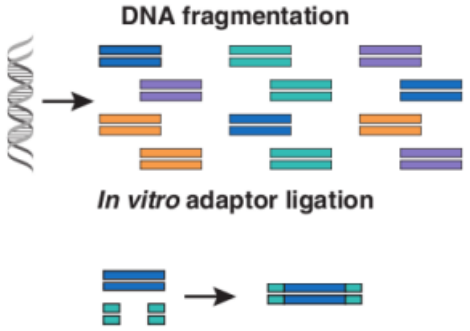
Basically, we add an A nucleotide at both ends of the fragments and we ligate (hybridization) an adaptor that has an overhang T. Then we use specific primers for the adaptors and we sequence the fragment.



Hence, we can create a genotec in a couple hours.

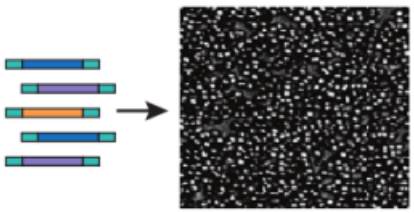
Then we just bind them to the sequencing device and obtain all the sequences.
As we said, we do rounds of adding nucleotides:

- First round we just add labeled A and make a picture
- Clean the unbound nucleotides
- Add labeled C and make a picture
- Clean…

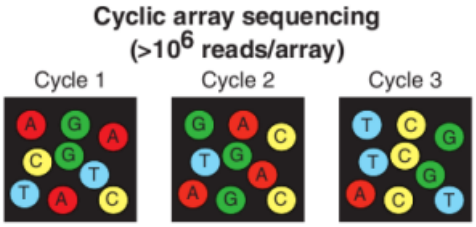


If we do 200 rounds, we will obtain sequences of length 50 (because we add 1 type of nucleotide each round).

Each time a nucleotide is incorporated, we get a signal. So, we will detect in which spot of the slide there is a signal each round (doing an image).
Finally we just need to stack all images.



Que pasa si hi ha dos nucleotids iguals seguits? En la mateixa ronda s'uniran els dos nucleotids i per tant veurem el doble de senyal?
We will detect an increased signal. But we just got one picture.
But since we detect the incorporation of one nucleotide in many sequences, we just do an average. The polymerase only adds one nucleotide.



NanoPore Single Molecule Sequencing

The sequence passes through a pore that detects each nucleotide. Each nanopore measures the electric current that flows through.

Each base that passes through the nanopore can be identified through the characteristic disruption it causes to the current in real-time.

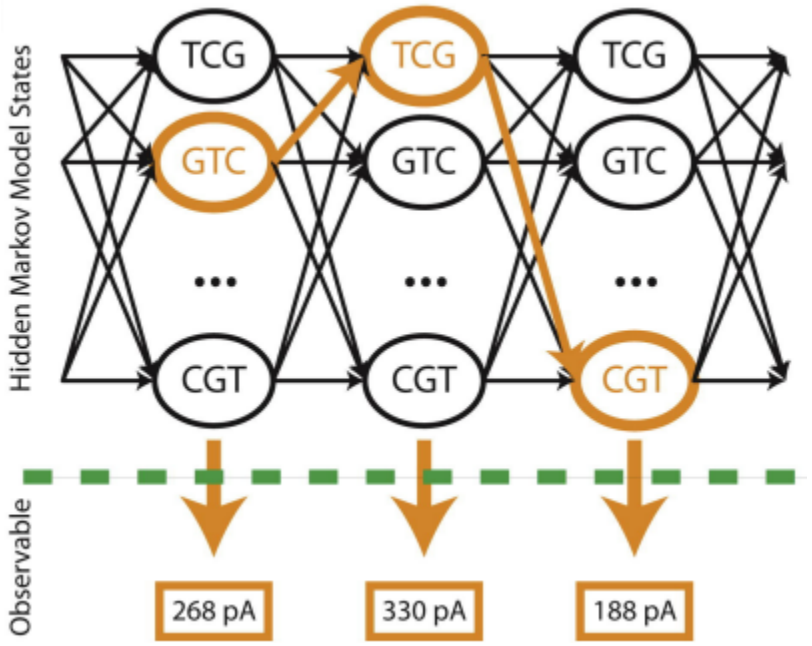
There is a lot of noise and sometimes it can be hard to interpete.
This technique is not based on detecting a single nucleotide but hexamers or octamers…
Each combination of nucleotides will produce a current variation that will be detected by the nanopores. So, we will be able to detect the composition of that k-mer.

How can we know that we have AGT and not TAG?
To know this, we will use a HMM.

In this example we have a HMM that has the observable states (current pics) and the hidden states that are the k-mers that could generate the same pic.

For the first k-mer, we give as input the current signal and it gives us all the possible k-mers. Then we give the next current signal and it will generate all the possible k-mers…

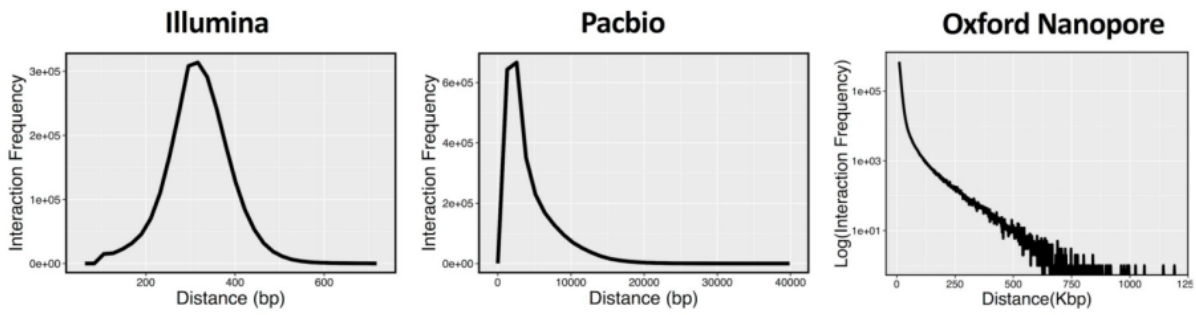
The thing is that there is an overlap between the k-mers. Thus, if the first k-mer is ATG, the next k-mer will start with TG-
At the end, the base caller is not just reporting the current changes but putting that into the probabilistic method and trying to find the optimal path through the different states that can emit a specific k-mer.



It is used in metagenomics to detect species, since it works in real-time.

Reads Length Distribution

From each technology, it is interesting to know which is its length distribution.



Applications of sequencing technologies

RNA-seq: Find the level of expression of some genes (transcripts)
Chip-Seq: Detect the fragments that are bound to a transcription factor. So it detects regulatory elements at genomic level.
ATAC-Seq: Analyze promoter regions

So, there are many variations of the experimental technique in order to capture the sequences that we want. So, when mapping on the genome we will obtain some sequences or others.

Errors in Sanger Trace Sequences

Remember that in Sanger we have a selection by length, so that the shorter sequences are detected first and the longer sequences are the last ones when we run them in an electrophoresis gel.

We expect that the errors in Sanger sequences accumulate at the beginning and at the end, because as we said, those sequences will run too fast or too slow when going through the detector (the peaks will not be equally distributed). Thus, the detection of every marked nucleotide will be a little bit desynchronized.

Medium length fragments will have a synchronized detection. Thus, if we have bad scores in these fragments, something has gone wrong during the experiment:

- There are 2 DNA molecules in the Sanger experiment (specially when there are 2 alleles). The signals of both molecules will mix.

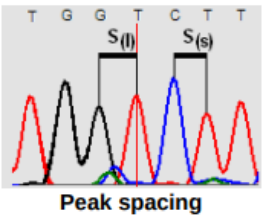
Often, the Sanger machine will provide 2 files:

- File that contains the nucleotide sequence (FASTA file)
- File that contains the quality score for each nucleotide

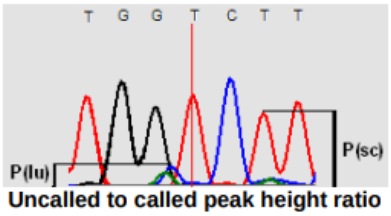
PHRED Score

This score is calculated taking into account different measures:

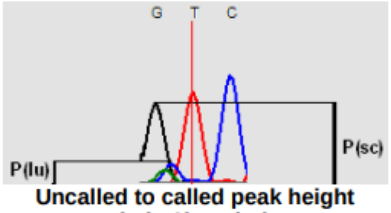
- Peak spacing: The peaks must be equally distributed through the electropherogram.
- The shape is normally distributed
- Difference between the top peak with respect to the worst peak.
- The different nucleotides must have a different height.
- Resolution: Distance from one peak and the next N (no nucleotide assigned)



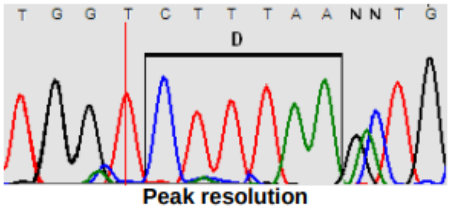
Peak spacing



Uncalled to called peak height ratio



Uncalled to called peak height ratio in 3bp window



Peak resolution

A combination of all these parameters makes a quality score (Q).

$$Q = -10 \log_{10} P$$

Since it is in logarithm 10 scale, the probability of incorrect base call is the following:

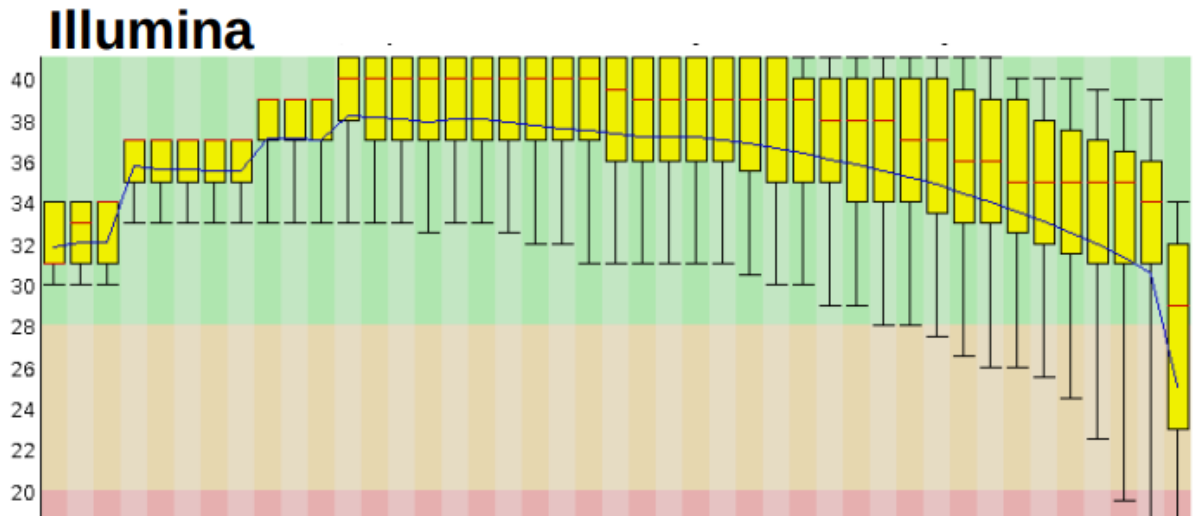
Phred Quality Score	Probability of incorrect base call	Base call accuracy
10	1 in 10	90%
20	1 in 100	99%
30	1 in 1,000	99.9%
40	1 in 10,000	99.99%
50	1 in 100,000	99.999%
60	1 in 1,000,000	99.9999%

Q20 is the threshold when cleaning and trimming sequences (remove low quality sequences). When trimming we must also take into consideration adaptor contamination.

Errors in NGS Read Sequences

The distribution of PHRED scores on NGS is different, so every sequence technology has its own biases:

- Sanger has biases in the 5' and 3' ends
- Illumina sequencing accumulates errors in the 3' end of the reads. As we move to the 3', we have more dispersion and the Q drops under 20.



For **Roche 454**, we have to remember that it is based on pyrosequencing.
Thus, if we incorporate more than one nucleotide, we get more light. Depending on the light, we can detect if 3, 4… nucleotides have been added.
The problem is that this saturates and we can not distinguish the addition of a lot of nucleotides.
These regions that have a lot of repeated nucleotides will have a low score.

In the case of the **Single Molecule** methods (pore based technologies, like nanopore), there are stochastic errors (random). We can solve this by aligning many sequences and detecting the errors.
If there is a nucleotide variant (not an error), we will see a higher frequency of that mutation.

FASTQ Record

Instead of using a FAST file for nucleotides and another FASTA for the scores, the FASTQ was designed:

- We have the nucleotides
- We have the scores translated into a series of symbols
    - Letters mean a good score

```
@HWI-EAS209_0006_FC706VJ:5:58:5894:21141#ATCACG/1
TTAATTAATAAATAATAGCTTAGATNTTACCTTNNNNNNNNNNTAGTTTCTTGAGATTTGTTGGGGGAGA
+HWI-EAS209_0006_FC706VJ:5:58:5894:21141#ATCACG/1
efcfffeefffcfddf`feed]`]_Ba_^__[YBBBBBBBBBBRTT\]][]dddd`ddd^dddadd^BBB
```

NGS Sequencing Biases

We have talked about the genome intrinsic biases:

- Genomes do not have uniform GC content, for example.

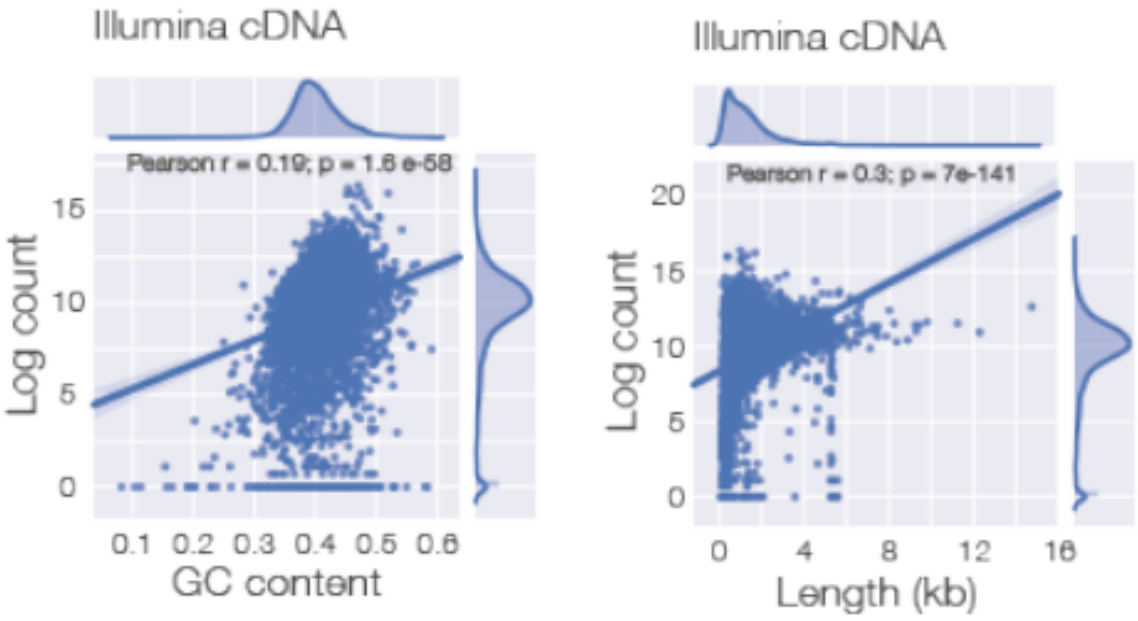
Those biases in the genome are affecting sequencing? In other words, are the sequencing technologies free of experimental biases?
One thing is having issues with the priming or with the enzymes, so I do not get enough fragments or the experiment goes wrong… But another thing is, what can be wrong depending on the sequences we give as input?
Yes, they do affect the sequencing.

So, each technology can sequence in a GC content limited range.
If a non model organism falls in an extreme value GC content, we will have a problem.
Because the sequencing techniques won't be able to sequence the whole genome (just a % of the genome, which has a GC content inside the range).

The abundance of the sequences can also introduce a bias.
In a RNA-Seq, maybe you have a gene that is expressed 1 million times more than another. Both transcripts (1 million transcripts vs 1 transcript) will compete with the polymerase reaction, with the enzyme, with all the chemicals in the kit…
As expected the 1 million transcripts will bind with the polymerase many more times.

Thus, we will have much better sequencing when we have a more expressed gene.

Depending on the GC content or the length, there is a bias.



Less is more…

For the single molecule experiments, we have less coverage but we have longer sequences (connectivity improved). This facilitates the assembly.
Even having less reads, it is possible to get similar accuracy assemblies because the length of the read can compensate for the lack of coverage.

4th Theoretical Class

Sequence Analysis

Prefix Trees or Tries

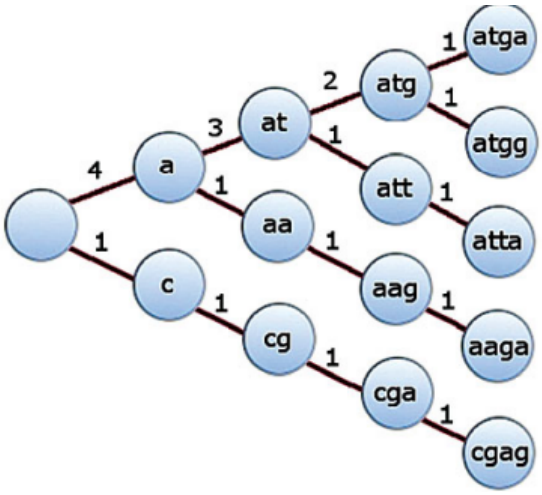
Trees are a subtype of graphs in which we have hierarchy. Meaning that there is a top level node, internal nodes and leaves.

We can use a data structure that allows us to store each of the symbols that define a string (prefix, suffix, k-mer, full DNA sequence…).

In the tries, we put the corresponding symbol on each level. So, the first symbol goes to the second level nodes , the second symbol of the string goes to the third level nodes…
At the end, we will have the leaves with the strings.

It is faster to look up data, it's more efficient than hashes…
Hash tables can have coalitions. But in prefix trees, since each branch corresponds to a unique symbol, the tree expands as we move to lower levels and there are no collisions.

Here we have 5 different k-mers indexed and, as we can see, each node can have a counter. So, it is easy to count how many different k-mers we have in our sequence and also how many times each of those k-mers appear in the query sequence.



Suffix trees are more efficient.

Counting K-mer Frequencies

When analyzing a sequence, apart from computing the GC content, we can also find how many k-mers of a given size our query sequence has.
So, we will count the frequencies of the k-mers.

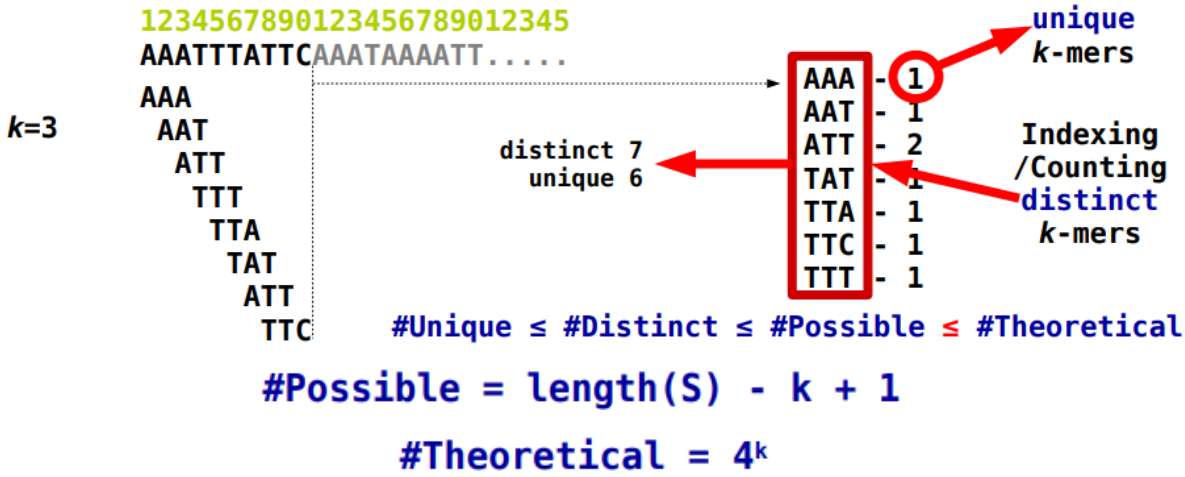
From a query sequence, we can define a k-mer size and run a running window (length 1, for example) to obtain how many times each k-mer appears.

If the k-mer is small, its frequency will be very high.
If we use a larger k-mer, the probability of finding them by chance will be smaller. Thus, if we find a high frequency, maybe it's a repetitive element.

How many times can we find a 3-mer on a sequence? Every 4^3 nucleotides we will find that 3-mer by chance.

To count k-mers it's easier to use prefix trees than suffix trees!

Unique k-mers are the ones that only appear once.



Possible = How many k-mers are in a sequence. In this case 10-3 +1 = 8
Theoretical = 64
Distinct = 7
Unique = 6

If we use larger k-mers, then the unique k-mers will be similar to the distinct k-mers.

K-mer Counting and Strands

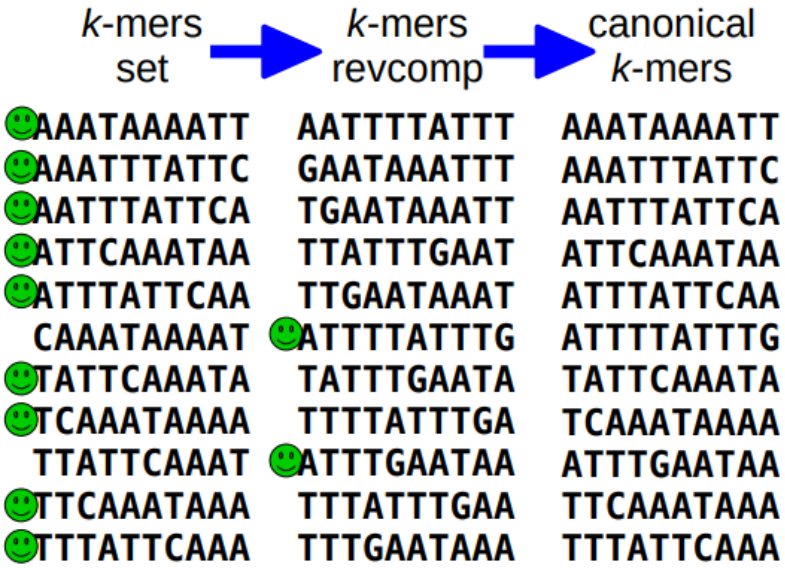
When we are counting k-mers, we are just counting the k-mers in one of the strands.

We can also count for the reverse complement sequence and the obtained k-mers will be different (the sequences will be different, they will be reverse complementary).
But, they correspond to the same element in the genome.
If I have a k-mer in the forward strand that is repeated, the reverse complement sequence of that k-mer will also be repeated (so it's like counting twice).

If we use both strands, we will obtain a more complete catalog.
We can apply a method that simplifies the k-mers we obtain from both strands into an average k-mer set known as canonical k-mers.

Pipeline:

- Obtain the k-mers from a given sequence.
- Make the reverse complementary of the k-mers (will have the same count as the forward k-mer). Not the reverse complement of the sequence and then compute again the k-mers (this will be slower. Because when we have the k-mers, we have only the uniques).
- Make the canonical k-mer set that corresponds to the k-mer that is alphabetically smaller.



K-mer Counters

We have a problem when counting k-mers. For a given size, some programs run out of memory, out of time…
They can not hold the data structure to store the counter for the different k-mers…

Just calculate how many k-mers of size 50 we can theoretically have. Assuming that each k-mer takes 1 bite, we don't have enough memory.

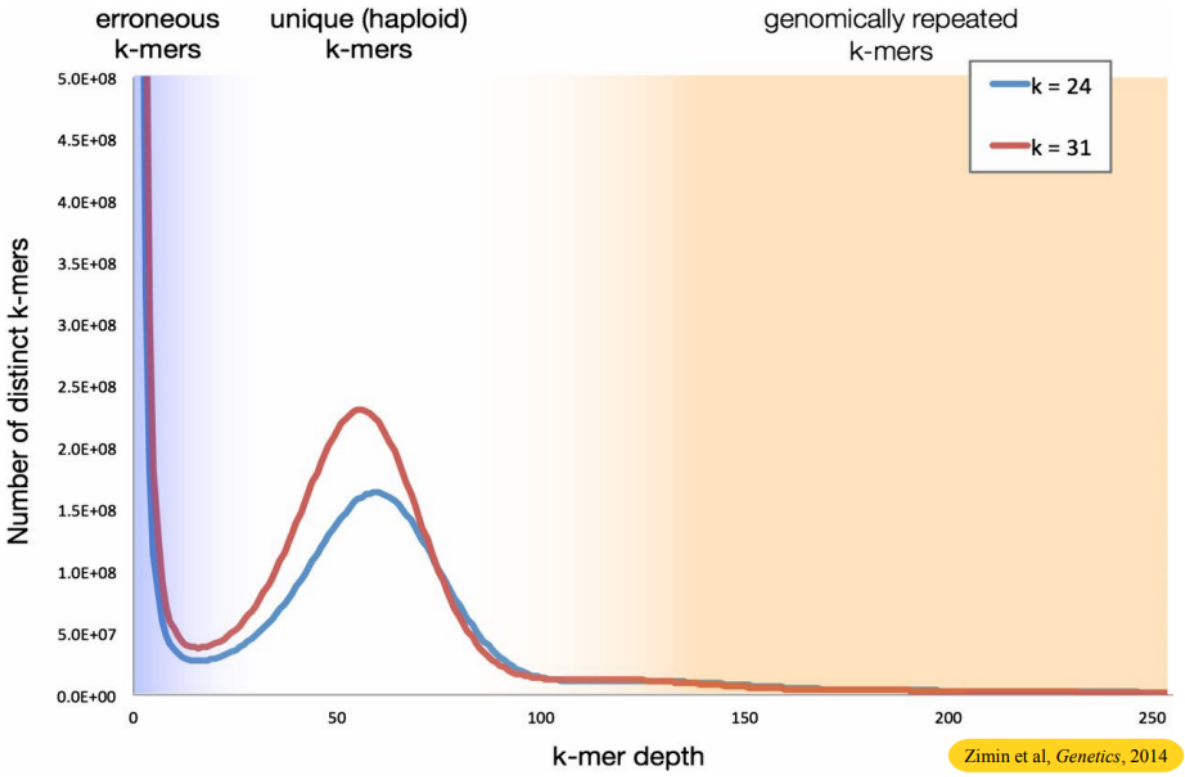
So it is important to improve the data structure that stores the information.

Which is the solution? One thing is the theoretical number and another one the possible number of k-mers.
If one query sequence has 1Gb, the sequence can generate as much as 1Gb of different k-mers. Assuming that each position has a different k-mer.

Distinct k-mers Depth

Why is it useful to count k-mers?
Here we have a distribution of the counts. This is not how many times a k-mer appears but how many times a k-mer returns a count of 1 or 2 or 3…

In the x-axis we have how many times we found k-mers one, 20 times, 50 times…
So, for X = 1, this corresponds to how many k-mers are unique.



Most of these types of distributions have this shape.
We can use this data to clean up the sequence dataset.
So, if we have reads obtained from sequencing, we can find which reads have the highest number of unique k-mers because those are probably sequencing errors.

Because when we are sequencing, we have a sequencing depth. So, when we run a sequencing experiment they tell us that we are going to obtain a 20x, for example. Meaning that on average we will obtain 20 copies of each read.

So, if we have 20 copies, we can not get a unique k-mer. We should obtain a count of 20 for each k-mer at least. The X axis of the graph should start at 20.
Thus, we must remove them!

In our case, it is likely that the sequencing depth is 50x. Because there is a pic in 50.
So, at 50 we find the unique k-mers and we should remove all the k-mers that have a lower depth.
This is for haploid, but if we had a diploid organism, the sequencing experiment would be 25x because we have 2 copies.

What can we interpret from the k-mers that appear 100 or 200 times? They are repeated and thus they may be repetitive regions.

Does it mean that the sequence itself was repetitive? Not necessarily, there can be artifacts. Maybe it's mitochondrial DNA.
So, if I am working with total DNA and I haven't removed mitochondrial DNA, I will have other pics that correspond to the k-mers that were produced from the mitochondrial fragments.

We must also take into consideration that the genome can be diploid and the mitochondrial DNA is always haploid.

So, in this case it's 50x and we know that due to the bacterial chromosome we have 100 times more copies in the sample. We will have some k-mers, derived from the mitochondrial genome, that will appear 50.000 times.

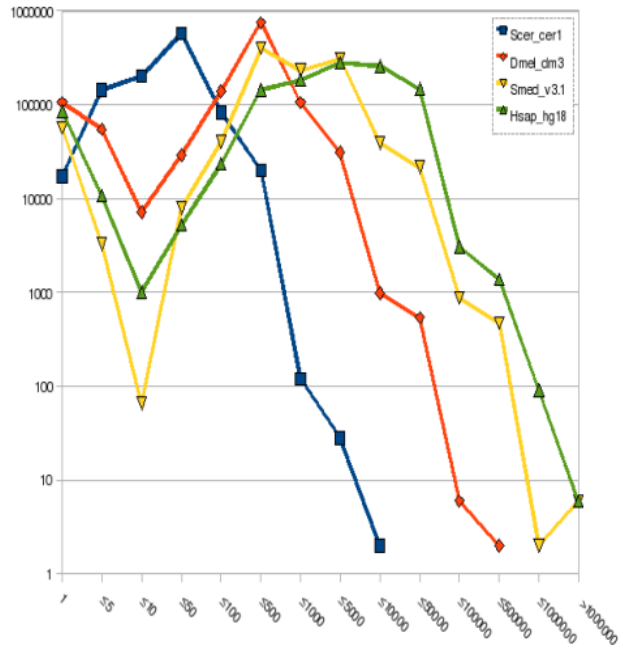
Despite using different k-mer sizes, we can see that the shape is similar. Even though:

- There are more unique k-mers if the size is larger (the red curve is shifted to the left)

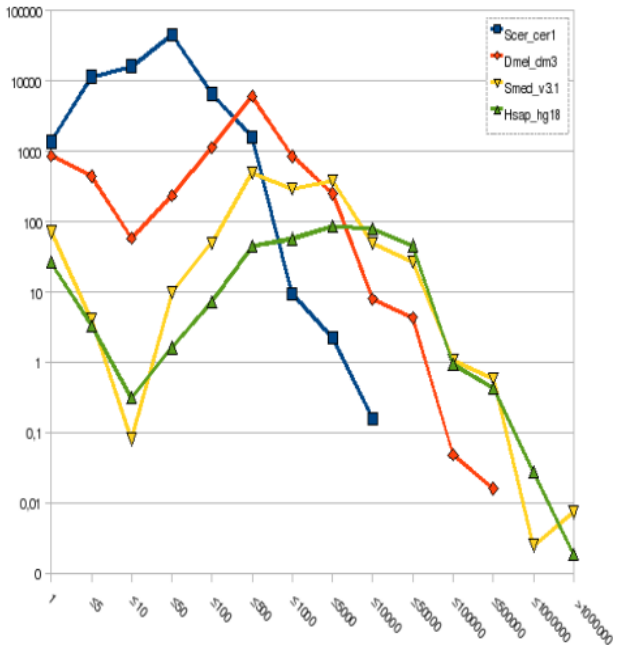
K-mers Depth on Different Genomes

We should expect that the larger the genome is, the more distinct k-mers we will have. So, it should have a tendency to go up.
We also have more counts on complex genomes. So, simpler genomes generate less different k-mers.
But, if we normalize by size, the trend is to accumulate counts to the right, which correspond to the repetitive elements.
We can see that the human genome has a lot of counts on k–mers that appear more than 10.000-50.000 times. Which is related to the C-value paradox (the size of the genome and the complexity are not correlated).

k=10bp



k=10bp, counts adjusted to genome size



Ultra frequent k-mers in Human Genome

Most of them are very low complexity sequences and they may be characteristic of regulatory elements. For example, the TATA box.
All genes need a TATA box to initiate transcription and, hence, if a genome has 2000 genes, there will be at least 2000 repetitions of the TATA box.

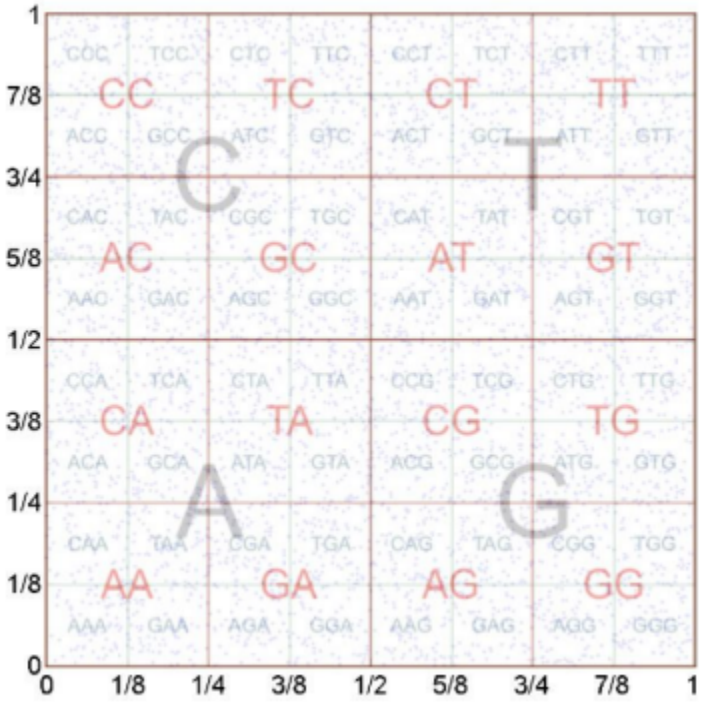
We will also see a lot of strong splice site signals.

Also, we can find specific repetitive elements like ALUS, SINES… that are species specific families of repetitive elements. So, the repeats are not shared across all the taxa.
As the genomes accumulate variation, the repetitive elements also evolve. Moreover, they are not as restricted as the genes to evolve and thus they will differ a lot across the species.

So, we can detect those repetitive elements by counting k-mers.

DNA Chaos Plot

Another approach to characterize the structure of a genome is using chaos plots.
We divide the plot into quadrants that represent a nucleotide.
Each quadrant is then divided into another quadrant…
So, we will have substrings in each square of the quadrant.



With this, even if we can not store or calculate all the possible distinct k-mers of a given length, we can plot them. Because all the possible k-mers fit in the plot.
If I have 10 sublevels and enough pixel resolution, I can plot all of them.

If I have a pixel for all the possible combinations of sequences, then we will have a full colored square. But, as we said, different genomes will have different composition and will produce different k-mers.

So, we are visualizing in a plot all the k-mers contained in a genome.
Here we do not see how many times a k-mer appears but if it appears.

It is a variant of the suffix tree. So, we can use the chaos plot to visualize the suffix tree.

Shannon Entropy

We can use some formulas that allow us to calculate how complex a sequence is.

$$H(X) = -\sum_{i=1}^{n} p_i \log_2(p_i)$$

p_i is the frequency of each of the symbols in the alphabet.

It can be taken as a measure of noise.
Entropy is probably a bad name, because it represents the expectation of finding or how probable it is that we get a given symbol of the alphabet (nucleotides or aa) when we are looking at the next symbol of a sequence.

If we have a random sequence, can we guess which will be the next character? Yes. All the characters have the same probability of appearing.

DNA sequence alphabet = 4 symbols/states $H_{DNA} = -4 \cdot (1/4) \times \log_2(1/4) = 2\ bits$

Protein sequence alphabet = 20 symbols/states $H_{AA} = -20 \cdot (1/20) \times \log_2(1/20) = 4.322\ bits$

If we are working with a low complexity sequence (AAAAAAAAA, for example), the next character will probably be an A. The Shannon Entropy will be small because there is small uncertainty.

So, we can use this measure of complexity to detect sequences that are unique, sequences that are random and sequences that are repetitive.

We can use this formula based on the number of symbols to estimate the maximum amount of information we can find on a given position:

- A position in a DNA sequence can hold up to 2 bits of information
- A position in a peptidic sequence can hold up to 4.322 bits of information

Why do we use “bits”? Because we are doing a logarithm in base 2. So, the base is binary and “bits” is the contraction of “binary digit”.
If we use a natural logarithm, we should use “nat”.
Also, it's not a fiscal information unit (like meters, for example).

How many computer bits do we need to store a symbol in the DNA alphabet? Just 2 bits.

How many computer bits do we need to store a symbol in the protein alphabet? 5 bits. We can not say 4.322 bits. So there is a bit of redundancy.

This means that when we are storing a sequence as a character, I am using a byte.
The nucleotide “A” is stored as 065 in ASCII code and this corresponds to 8 bits (01000001) or 1 byte.
Thus, we are wasting 6 bits to store a single nucleotide position. But, if I use the bits to store the information about the sequence, in a byte I can store 4 characters.
Hence, we are compressing up to 4 times the information. So, complexity also allows to apply compression algorithms to the data.

Which is the real information content (I) of a position that has already a defined nucleotide? When we are modeling transcription factor binding sites, we have different signals aligned together and we can find frequencies on a given position of the model.
So, how much information does that position of the model have?
It depends on the frequencies.

From the Shannon Entropy we can derive the **information content (I)**, which is weighted by the ratio of the observed frequencies and the expected frequencies of the background model, which is normally the random model.

$$I(X) = -\sum_{i=1}^{n} p_i \log_2(p_i / q_i)$$

$$X = \{x_1, x_2, \ldots, x_n\}$$
$$p_i = P_{observed}(x_i)$$
$$q_i = P_{expected}(x_i)$$

So, with this, we can estimate the real number of bits a given position of a model has.

The information content focuses on a position in a model

We can also extend the entropy to this other complexity entropy (**CE**) that takes into account the length of the sequence.

$$CE_S = -\sum_{i=1}^{k} \left(\frac{n_i}{N}\right) \cdot \log_k \left(\frac{n_i}{N}\right)$$

$$x_i \in \{\alpha_1, \alpha_2, \ldots, \alpha_k\}$$
$$S = \{x_1, x_2, \ldots, x_N\}$$

Orlov i Potapov, 2004

This allows us to compare different sequences.

We have another complexity measure, the **"linguistic Trifonov complexity" (LC)** that is based on k-mer composition. It takes into account different k-mer sizes at the same time. For a given sequence we count how many k-mers of the possible k-mers appear.

ACGGGAAGCTGATTCCA					
U_1	A	C	G	T	4/4
U_2	CA CC CG CT	AA AC AG AT	GA GC GG ~~GT~~	~~TA~~ TC TG TT	14/16
$CL_2 = U_1 \cdot U_2 = 4/4 \cdot 14/16 = 14/16$					

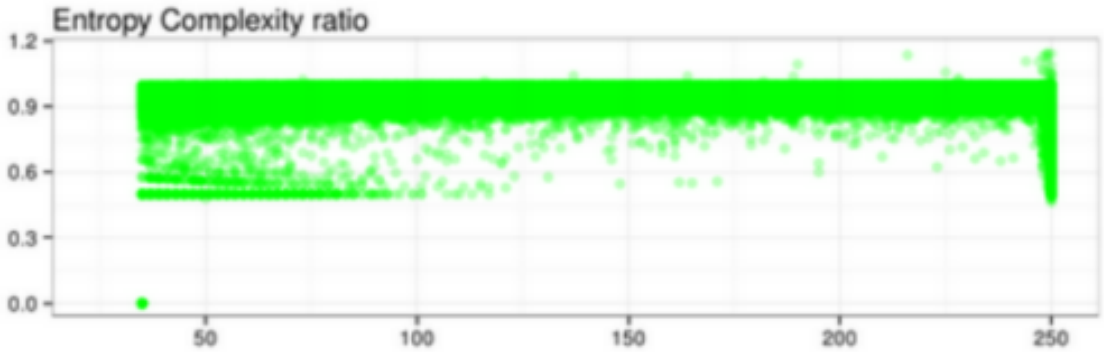
Here there are all the 1-mers, there are 14 out of the 16 2-mers.
We compute CL of level 2 by doing the product of 4/4 and 14/16.

So, it's the product of all the levels!

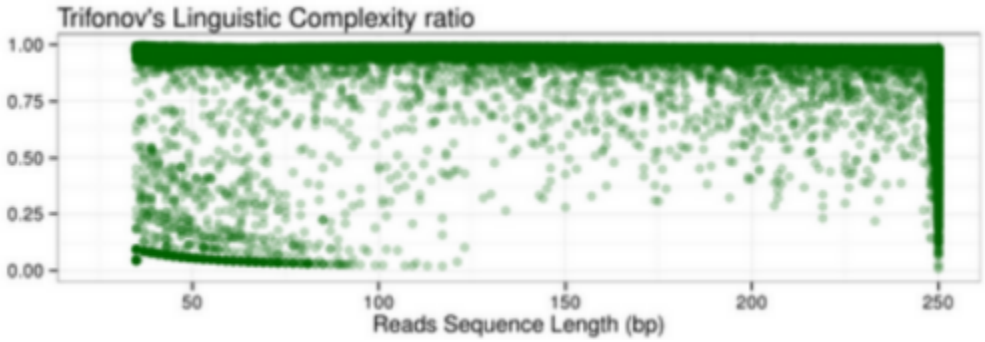
Plot Entropy Complexity ratio

The X-axis corresponds to the length of the sequence and the Y-axis corresponds to the complexity of the sequence.
We can see that many sequences are on top (they are complex) but there are also some sequences at the beginning that are not complex (also at the very end).
These are probably repetitive elements that can be removed.

So, we can define a threshold of 0.8 and remove all the low complexity sequences.



Here we can find the Trifonov's Linguistic Complexity ratio that produces really similar results.



Compression Algorithms

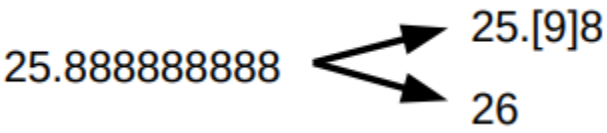
We have said that we can use the number of bits that we require to store the information for a given position and for a given alphabet.
So instead of using 1 byte, we just use 2 bits for DNA nucleotides.

If a sequence is highly repetitive, it will be much easier to compress it.

We must understand the difference between 2 types of compression algorithms (we will talk about the loss of information). If we are going to store a nucleotide sequence in a compressed form, we should be able to retrieve back the original sequence and, hence, we should not lose any information when compressing.

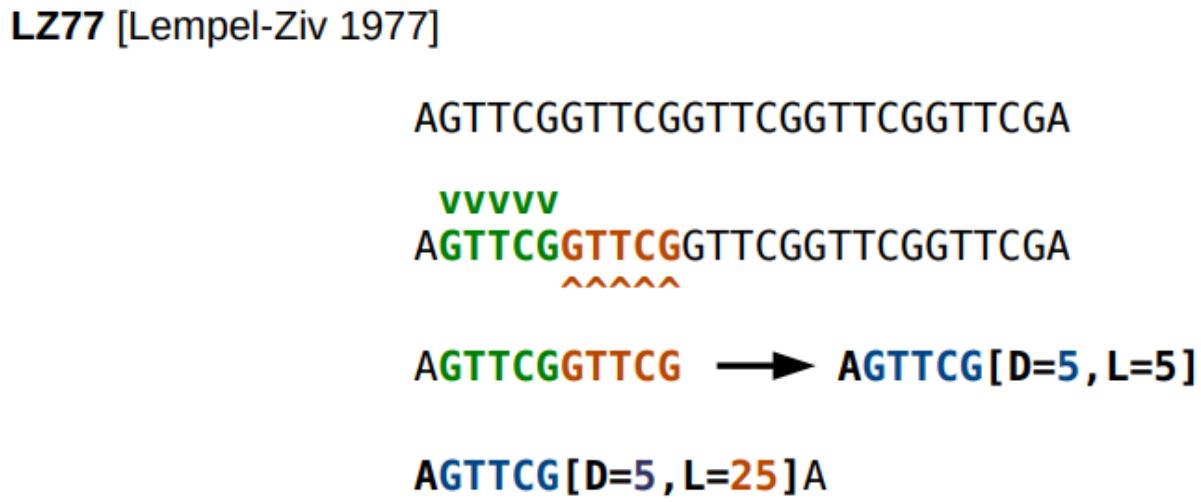
- **Lossless** compression algorithms: They do not lose any information
- **Lossy** compression algorithms: They lose an accepted amount of information

For example, we can store the following number in 2 different ways



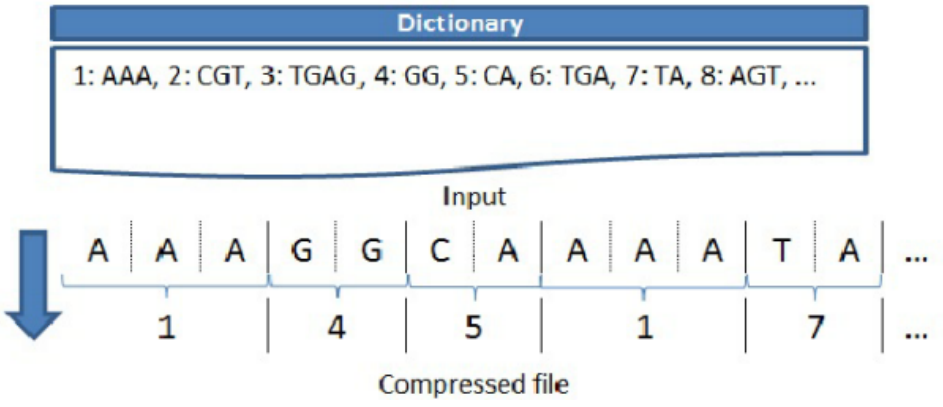
The first way says that there are nine 8 after the decimal point and the second one rounds to 26. As we can see, the second option compresses the data better but we lose information and therefore, it can not be decoded back to the original number.

One of the first lossless compression algorithms is the LZ77



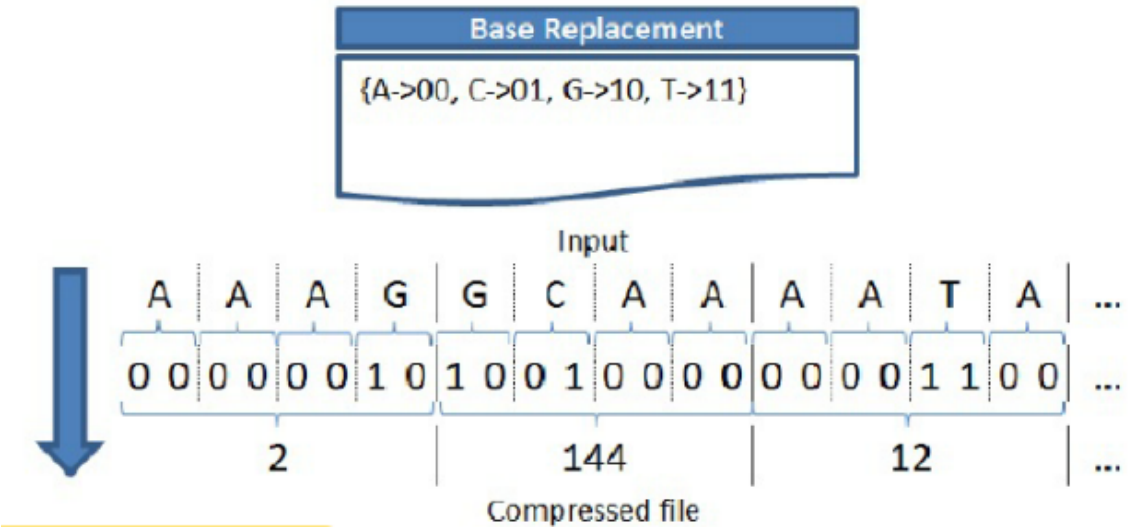
Different Sequence Compression Approaches

We can use dictionaries to store different sequences (value) linked to a number (key).



So, instead of using 3 bytes to store (AAA), we use 1 byte to store the number.
At the end we have a string of numbers.
It is important to attach the dictionary to the end of the file so that you can decode the string of numbers and obtain the original sequence.

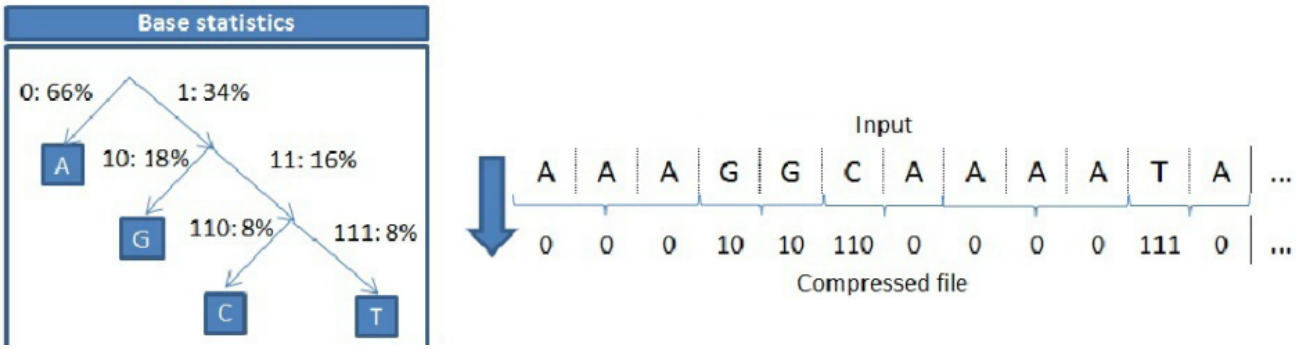
We can also use “replacement”. We can use 2 bits to encode each nucleotide and, hence we are compressing 4 times more at least.



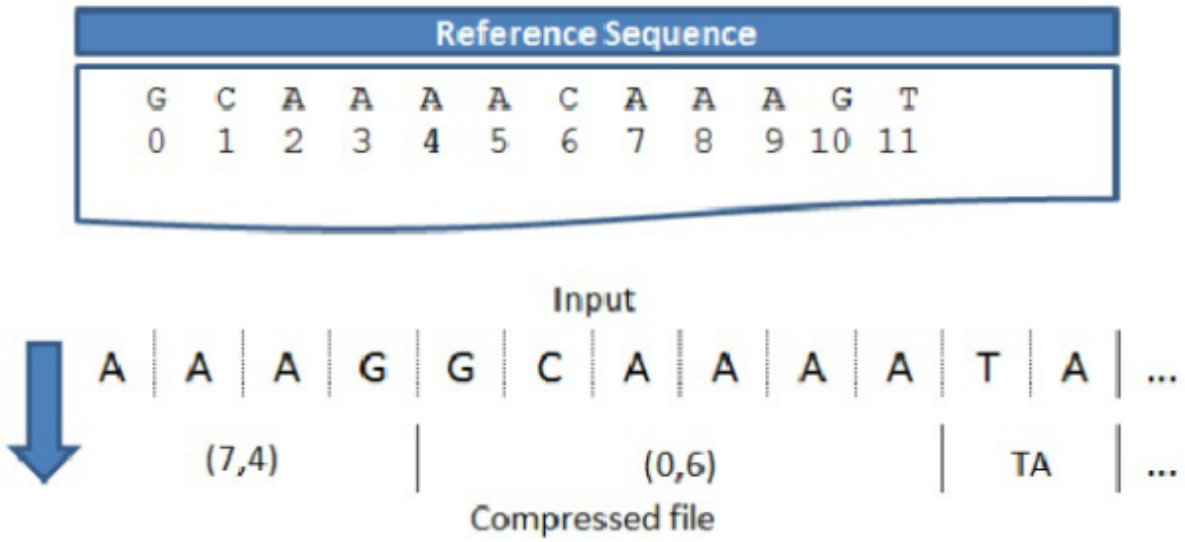
There is another approach that consists in taking advantage of the frequencies. If A appears most of the time, I can use a dictionary that has the smallest value to define A.
For example, A = 0 (we only use 1 bit).
But if we only use 1 bit, we can also encode 2 characters (0 or 1).

We just add an extra bit for the characters that are less likely to appear:

- A = 0
- G = 10
- C = 110
- T = 111



Finally, we can use a **reference sequence**.
We store the starting point of the reference sequence and the length.



From each algorithm we can get the % of compression for each sequence.
We can use this % to compare those sequences and define which have a lower complexity.

Because, as we said before, low complexity sequences are easier to compress.

It is important to store the data as much compressed as possible because:

- It is going to use less space
- The programs should be able to read files that are compressed. Thus, it will need much less time to compute the calculations that the program performs. The program reads the data faster.

5th Theoretical Class

Repetitive Sequences

As mentioned before, complexity measures can help us to detect repetitive regions (low complexity regions). Also, removing these regions is one of the first steps when we are annotating a genome.

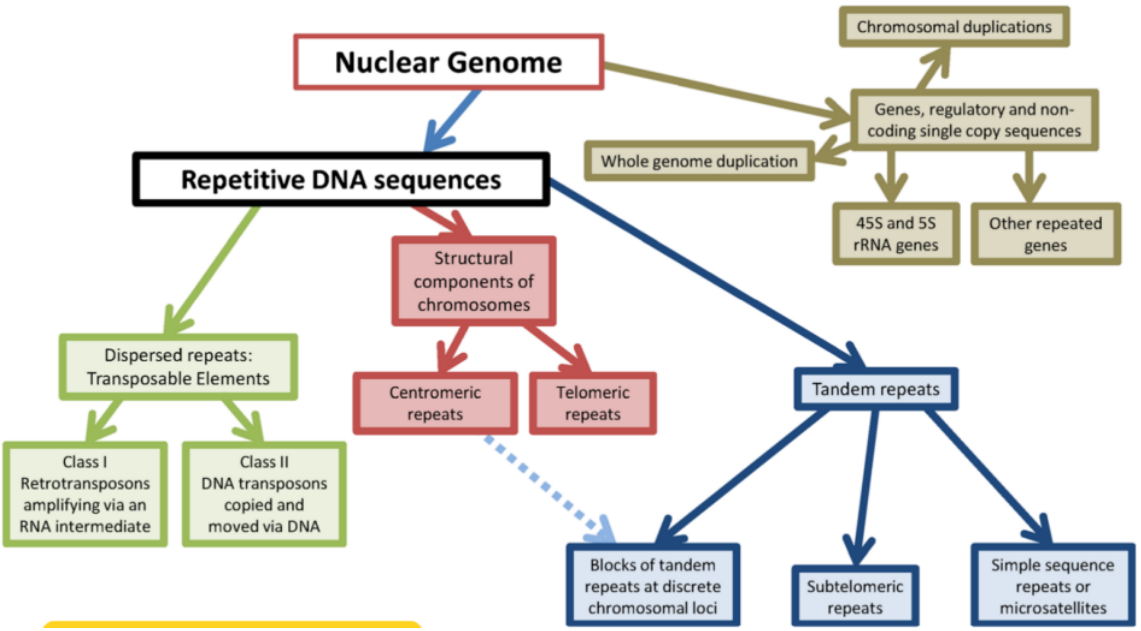
Note that 50% of the human genome are repetitive elements.

There are many repetitive regions:

- Centromeric repeats (important for chromosome structure)
- Telomeric repeats (important for chromosome structure)
- Tandem repeats (simple repeats or microsatelites)
- Transposons (Class I and II). They are known as selfish DNA because they have the capability of autoinserting into the genome. They have specific repetitive elements that facilitate the insertion into the genome and they can jump.
  Knowing these repetitive elements allows us to detect the transposons.

We can also have whole genome duplications and, thus, we have a double amount of copies. Genes can also be repeated (gene expansion). For example olfactory sensors in dogs.

Genes that are important for the translation machinery are also repeated in tandem and, thus they are transcribed at a higher rate (ribosomal genes).



Masking Repetitive Sequences

RepeatMaker: Uses a previously compiled library of repeat families.

BLAST Algorithm: Extension

BLAST passes a series of filters to remove low complexity regions.

Unit 3. Sequence assembly

The BioEarth genome project is the most ambitious project since it aims to obtain all the genomes of as many organisms as possible (at least 1 Million).

If we know the genome we can know the proteins expressed and use them for biotechnological purposes.

From tissue to Single Cell Genomes

There are different technologies to extract the DNA or RNA that is at single cell level.
So, we can detect when and where the transcript was found (organ, tissue…).

Why is it important to obtain the DNA or RNA of each of the individual cells of a tumor?
Because we can just create the phylogeny of the cell types and find which set of mutations define each type of cell and also know the state of the tumor.

Knowing the mutations we can know if the tumor is going to expand, and know the optimal treatment… We can also know which is the primary cancer (if the patient has more than one).
Explanation: A patient can have lung cancer (secondary) that comes from kidney cancer (primary). If we don't detect this, we may just treat the secondary cancer and not the primary. So, the phylogeny of the cells of the tumor allows us to detect the primary, secondary… tumors.

Tumor cells can become resistant to the treatments or drugs used against them.

We have the expression levels of the genes across different time points. So, we can reconstruct the expression of each gene with respect to the time and cell type.
Velocity plots merge information about how the expression of the gene changes with the cell cluster. So, we can see how the cell clusters evolve from a pluripotential to a specialized cell, for example.

Overall Assembly Summary

Step 1

We have the reads from a sequencing experiment and we will assemble them to form the contigs.
A contig is a contiguous sequence made of overlapping reads (so we have support for each position, because the reads overlap).
There is a 3' - 5' overlap (when a read ends, it starts a new one that was previously overlapped).

But, we normally don't have enough reads to make a unique contig. So, in some positions of the genome we do not have reads and, hence, there are some unsolved gaps.

We assume that we have a 20x experiment (we should have 20 overlapping reads per position). **Why do we not have 20 reads of every position?**

- This is because there are some regions that are hard to sequence because of the GC content bias (regions that are below 10% or above 80% of GC), for example.

- Another reason could be that when making the library, we are selecting fragments of a given size (optimal size for the sequencing device, 200-400pb).

  If we have a mechanical process like sonication or a chemical process that breaks the genome, we expect to get fragments of similar sizes, but due to repetitive elements or the composition of the sequence, we may get fragments that are too small or too big.
  So, when filtering the fragments we lose those regions.

In light of the above, GC content bias and the process of making the library affect the connectivity of the assembly.

The most difficult regions to obtain reads are centromeric and telomeric regions (specially when using Illumina).
So, depending on the coverage, we may get more or less gaps.

Step 2:

If we have further information, we can combine the contigs and map them into the chromosomal locations.

If we use a large insert reads library, we can use them to correctly place the contigs (sort them and also in the correct orientation) and make the scaffolds.
This is known as Scaffolding.

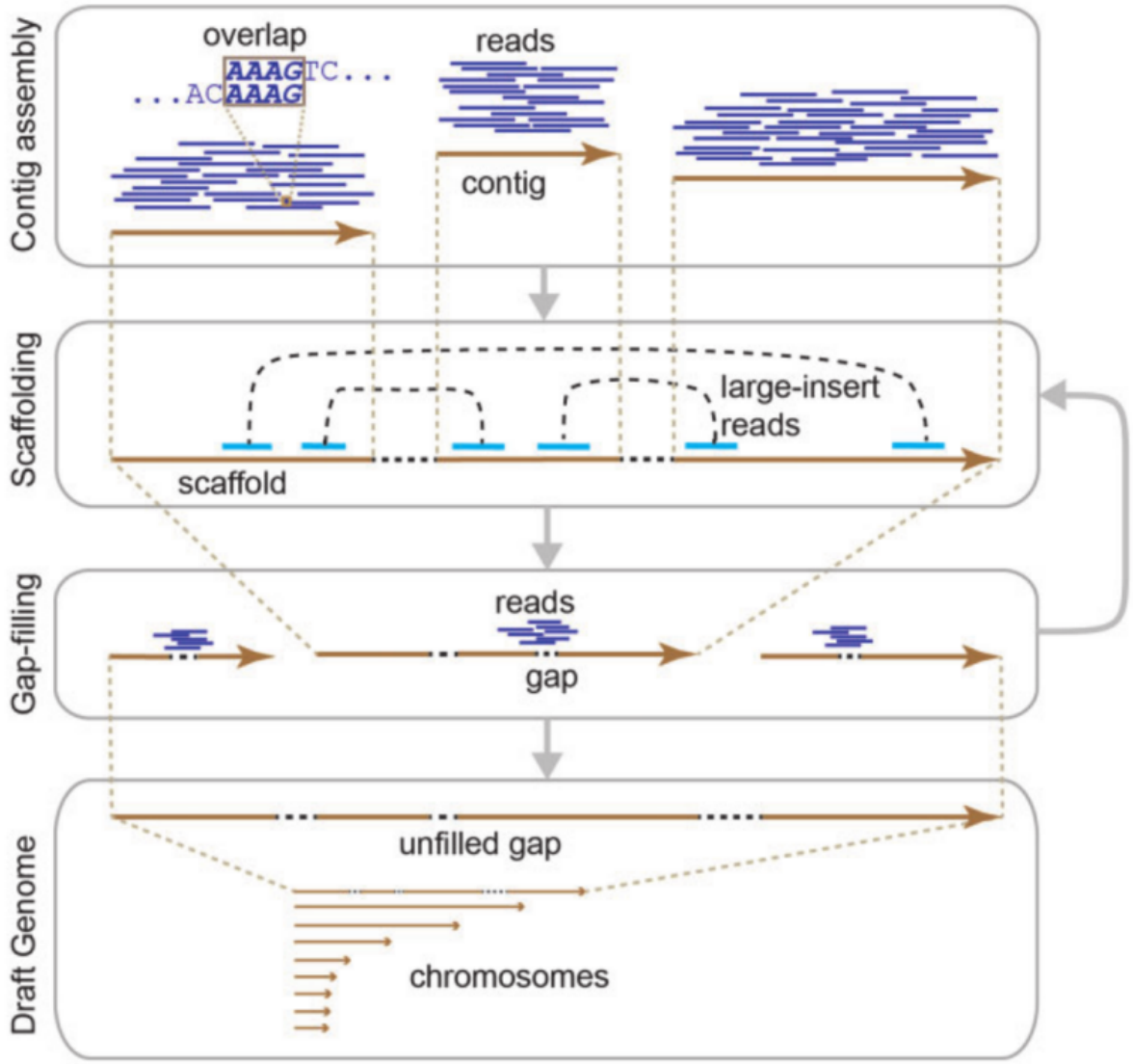
So, the short reads provide **contiguity** and large insert reads provide **connectivity**.

This is why we should have different size reads.

Step 3:
We have detected some gaps, and we know the sequences that flank those gaps.
Thus, I can design primers to sequence the gaps by Sanger (for example).

Once we have those sequences we can fill the gaps and obtain a contiguous sequence for the scaffold.

Step 4:
We can use further experimental evidence to anchor the scaffolds to the corresponding chromosome locations (i.e using physical maps based on recombination/ligation distances, optical sequencing, FISH barcoding, etc).



We can also use physical maps to make the scaffolds.

Sequencing libraries

Depending on how we prepare the sequencing libraries, we get different types of reads:

- **Single end reads:** Each read I obtain for either of the strands are independent. They come from a single sequencing run.
- **Paired end reads:** They come from a full sequencing run. Enables both ends of the DNA fragment to be sequenced. Because the distance between each paired read is known, alignment algorithms can use this information to map the reads over repetitive regions more precisely.
- **Mate pair reads:** It's a specific type of paired end read. But the insert size is really large and the orientation is different.

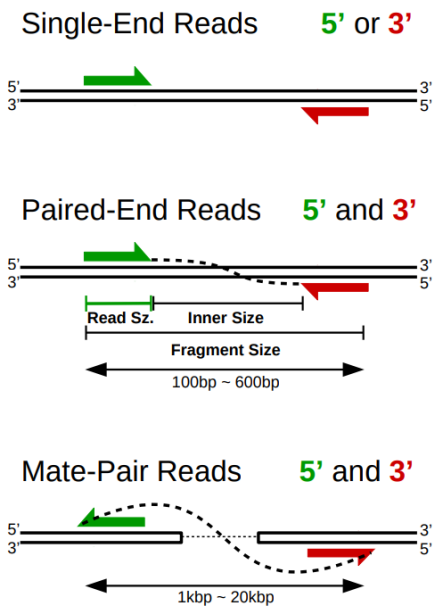
How is it possible to sequence a fragment of 1kb if the sequencer can only sequence fragments of 500pb? We can not put into the sequencer the fragment of 1 kb because it wont work. We have to make a special library.

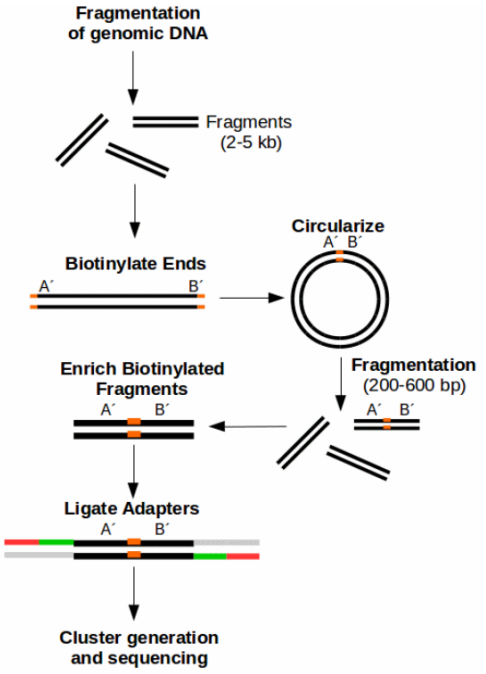
We make a single or pair ends library by simply adding the sequencing adaptors after filtering by the length.

We will have to go through an intermediate step. We add a marker (biotin) that is used to select the molecules that have biotin attached.

We circularize the molecule of 2kb by ligating the biotinylated ends.
I break at random the circularized molecule and select the enriched biotinylated fragments (200-600 bp). Now I can sequence this fragment since it has an optimal length.

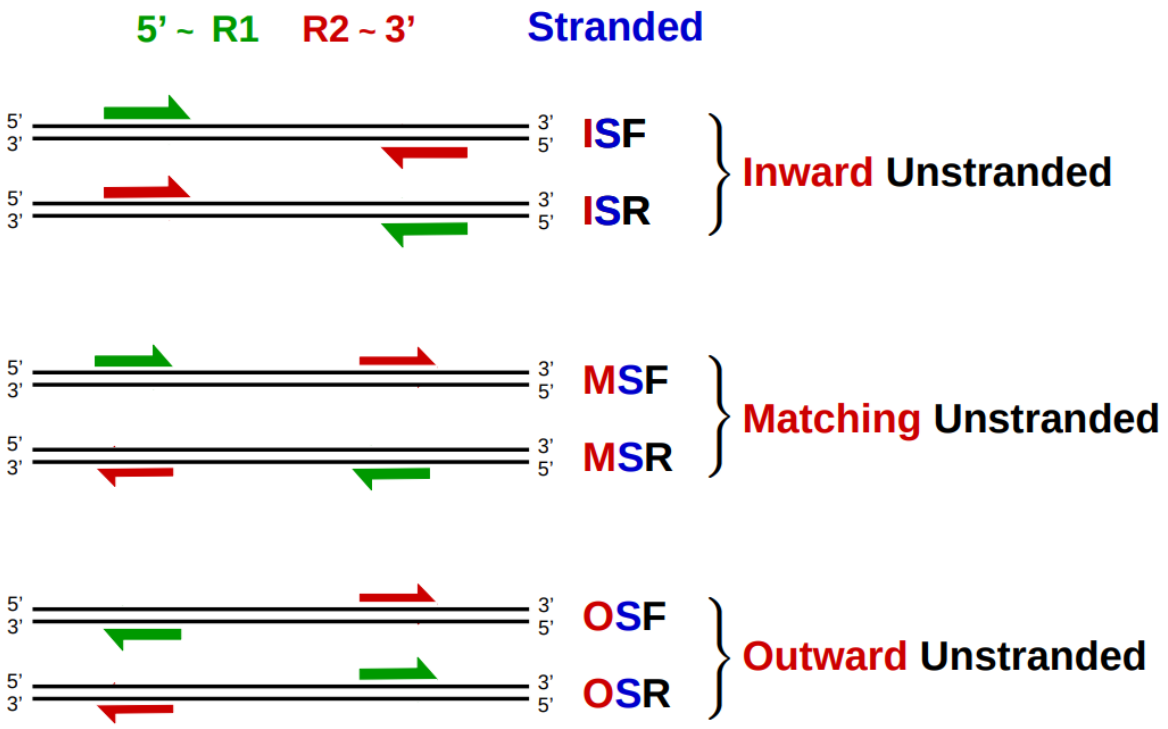
We add the sequencer adapters and as a result we obtain a paired end. Because I am sequencing a short fragment but the ends came from the ends of the circularized molecule.



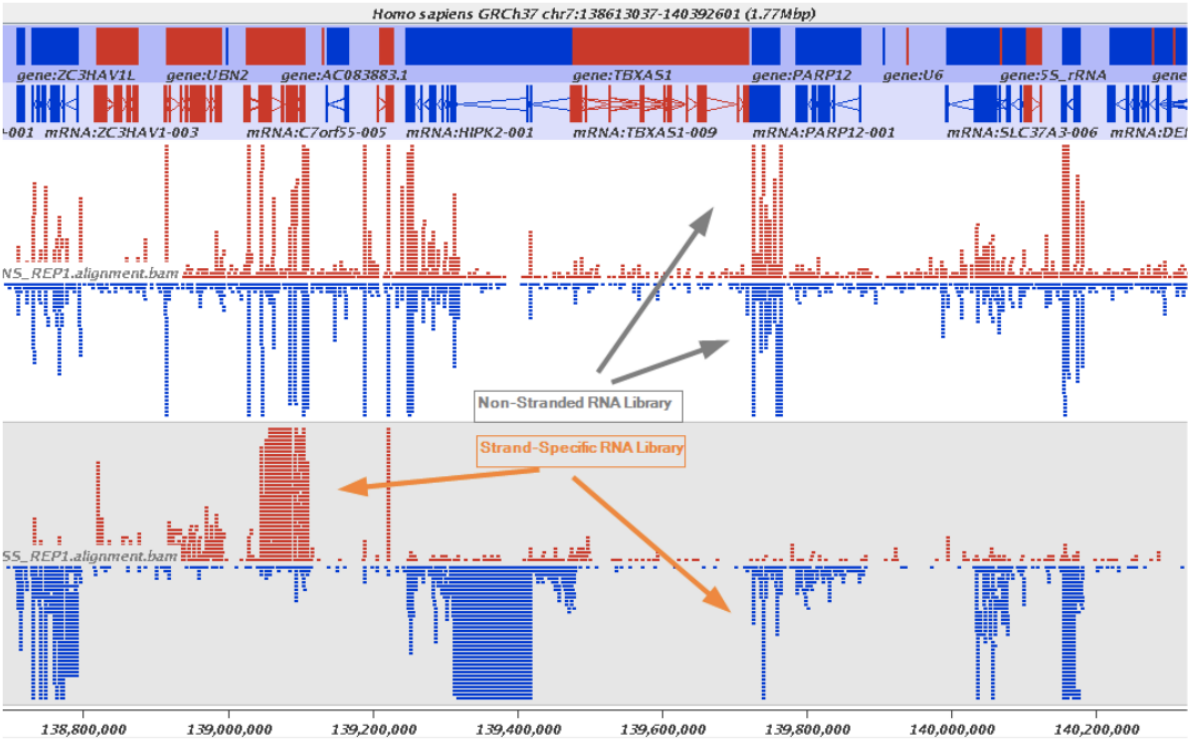


Fragment library types

Unstranded: Will give us



Stranded/Unstranded Fragment Libs



Paired-end TAGS

The technology can be used to explore the genome at different levels.

The DNA has different stages of packaging:

- Open regions that are being transcribed (contain the genes that are being transcribed)
- Nucleosomes that contain 100 nucleotides surrounding them (making a loop). This form a chain of pearls (like a necklace)
- There are other proteins that pack together loops and make regions closer (enhancers)

There are many things that we can analyze:

- At Chromosome level:
    - Delitions
    - Insertions
    - Inversions
    - Translocations
- At single nucleotide level:
    - SNPs

We can use the Paired-end TAGS (PET) to:

- **RNA-PET** is a technology that finds sequences of the transcripts and we can later map them to the genome.
  We can also circularize the transcripts and find the starting and end sequence of the transcript. So, with the paired ends or the single ends we can get the coverage of the exons and with the RNA-PET we can obtain the starting and ending position of every single transcript of the genome.
- **ChIP-PET** is a technology that finds the regions that define pics associated with transcription factors. So, we can use it to detect the transcription factor binding sites, regulatory elements, enhancers…
- **ChIA-PET** allows us to detect long range interactions
- **DNA-PET** allows us to characterize insertions, deletions or translocations at chromosome scale. So, I can see whole chromosome inversions (not a single gene deletion) or deletions of Megabases.

Cleaning NGS Read Sequences

We know that there are sequencing errors and other problems. So, we can just filter out by quality score. The standard quality score is q20, so anything that is below that threshold can be removed.

But there can be other things happening to our dataset.

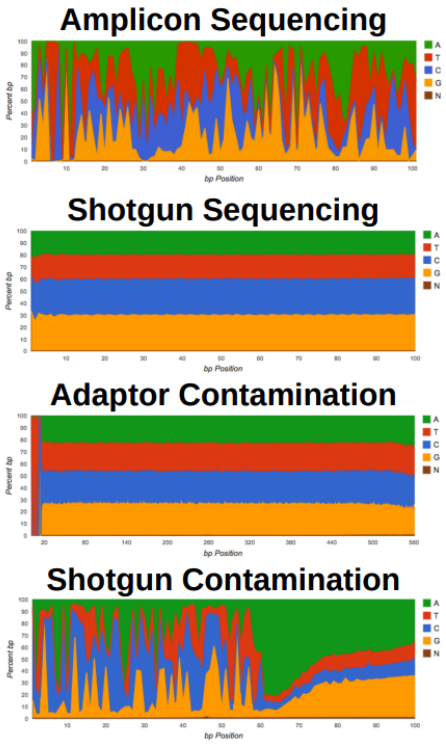
In these plots, we are aligning by position all the reads of our assembly experiment. So, we are looking at the composition of A, T, C and G for every single position.

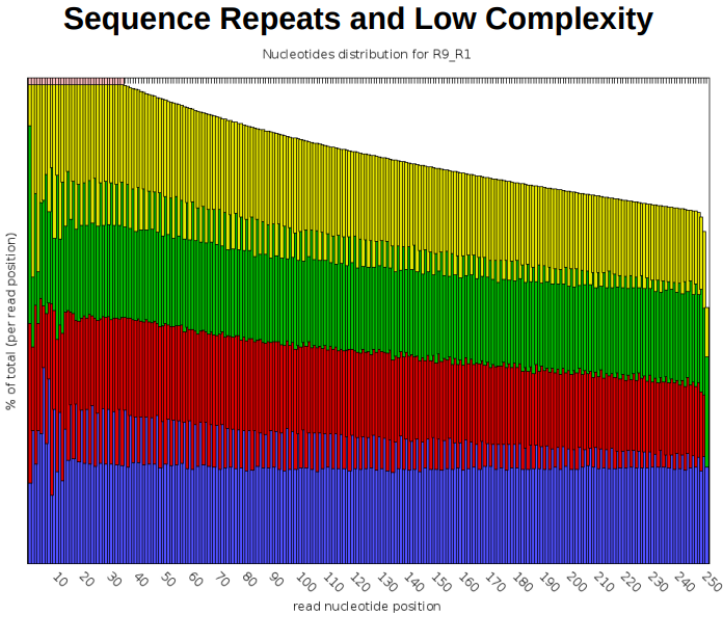
If we **sequence amplicons** (a specific region of the region), we expect that all the reads have more or less the same composition. So, we obtain strong peaks.

When doing a **shotgun sequencing** experiment, we are breaking at random the genome. Thus, if we align all the reads, we should expect the frequency of each nucleotide in each position of the read.

In the third plot we can see some peaks at the beginning of a shotgun sequencing experiment. This is because the sequencing reads still have some part of the adaptors. So, some part of the adaptors has been included in the sequence.
Maybe the insert was too short and we are sequencing part of the adaptors. Thus, we will have to trim this region.

In the last plot we have a mix of shotgun contamination with adaptors.





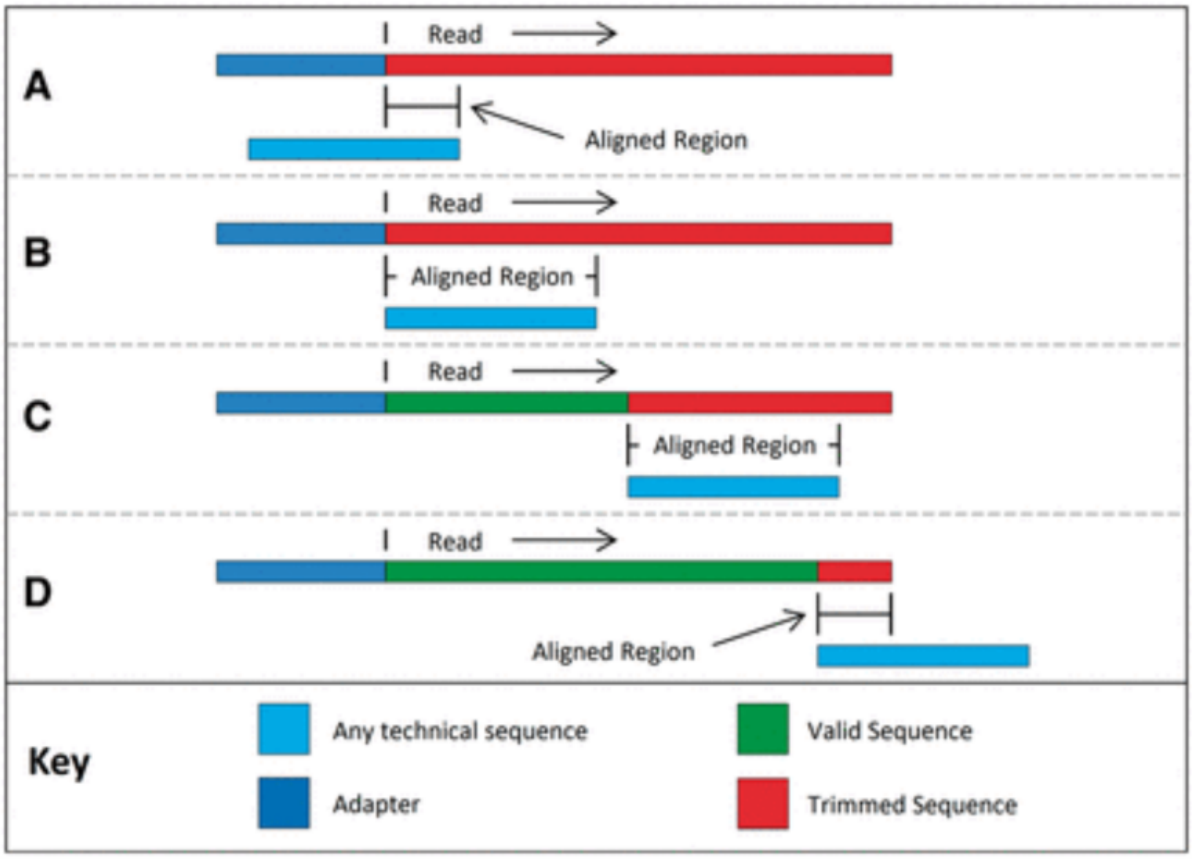
The big plot looks fine, since it looks like we have the same amount of G and C.
Note that we have longer and shorter sequences (we have less longer sequences).

There is something special. There are some types of peaks.
This is due to the repetitive elements. There are some nucleotides that appear more frequently than others.

Reads Preprocessing: Cleaning and Trimming

When we are sequencing the read, we have the adaptor and the read attached. So, we are getting the complementary sequence of the read during the illumina experiment.

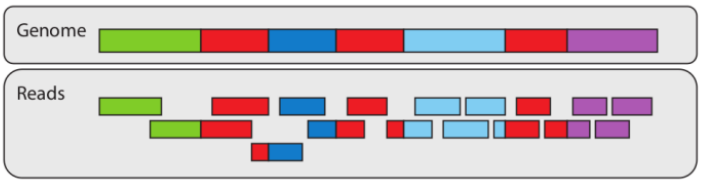
no entenc… (1:55)



6th Theoretical Class

Different Assembly Approaches

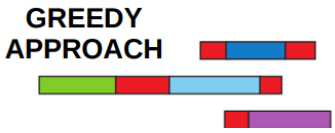
The genomic sequence is broken into reads at random using sonication or enzymes. There will be regions with biases that will affect the fragments we get. So, the reads will be the fragments that we were able to sequence.



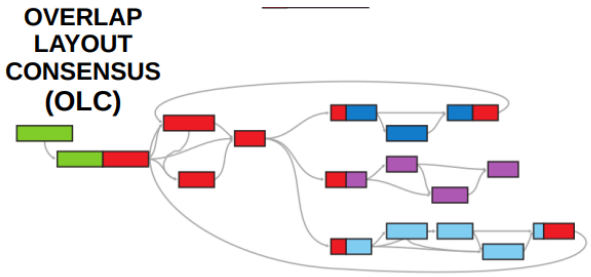
In the schema, the red parts correspond to repetitive elements and they can cause some problems during the assembly.

There are 4 assembly categories:

- Based on the **greedy approach:** Fast methods but the assumptions that they consider can lead to a wrong answer. They give us a solution but it is not the best one.
  They are based on aligning the 3' and 5' of the sequences and they evolved into the next approach.



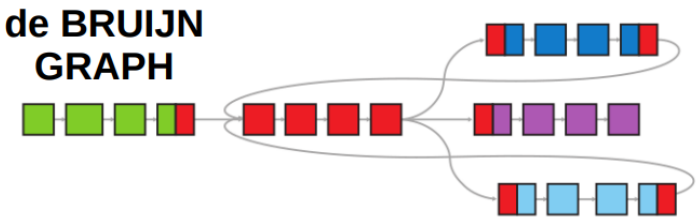
- **Overlap Layout Consensus (OLC):** Graph based approach in which we model the alignments into a graph. The nodes are the reads and edges correspond to alignments between them.
  So, we are projecting all the alignments and the correct solution will be finding the optimal and longest path through that graph.



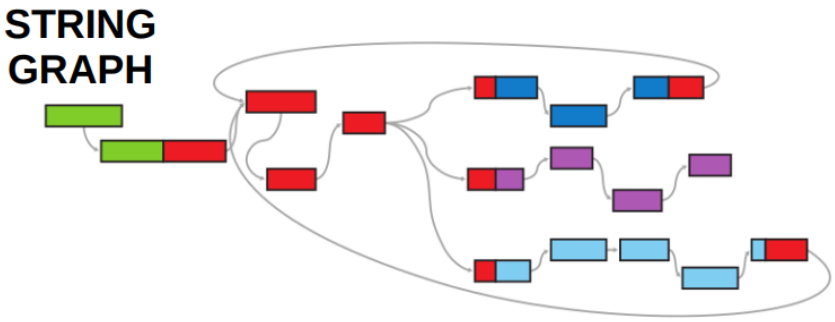
- **De Bruijn Graph:** Modern assemblers, instead of looking for alignments (which is inefficient because we have to align all reads against all reads and there can be spurious alignments), calculate the k-mers for each of the reads. Then if 2 reads share the same set of k-mers, in practice it is like they are aligned.
  If one read has 5 k-mers at the end and another read has the same 5 k-mers at the beginning, they will align.
  So, we just take all the k-mers of a fixed size (the reason why all the boxes have the same size) that can be obtained from the set of reads and we can easily know how they connect by looking at the overlap between the k-mers by shifting one position each time.

We will just align all the k-mers of a given size and project it into a graph.
Here we have a lot of redundancy, because we have several copies of the same molecule (look at the reads of the page before, they are overlapped and thus there are many copies).
If we have a sequencing experiment of 50x, we will have a huge set of nodes in the graph. The solution is to use the unique k-mers (remove redundancy) and, hence, the graph is much simpler and easier to interpret.
It is very useful for large genomes because we simplify a lot.

Note: The edges represent the overlap



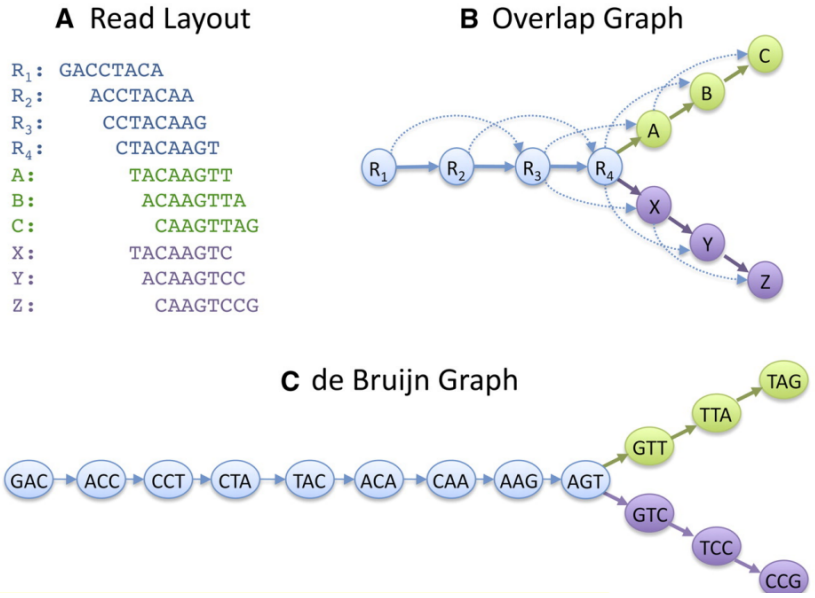
- **String Graph:** It's a simplification of "de Bruijn graph". We can compact the single paths into strings. We collapse k-mers.
  Very useful for large genomes.



Assembly Graphs

As mentioned, there is the aligned (Greedy and OLC) and kmer based (Bruijn and String) approaches.

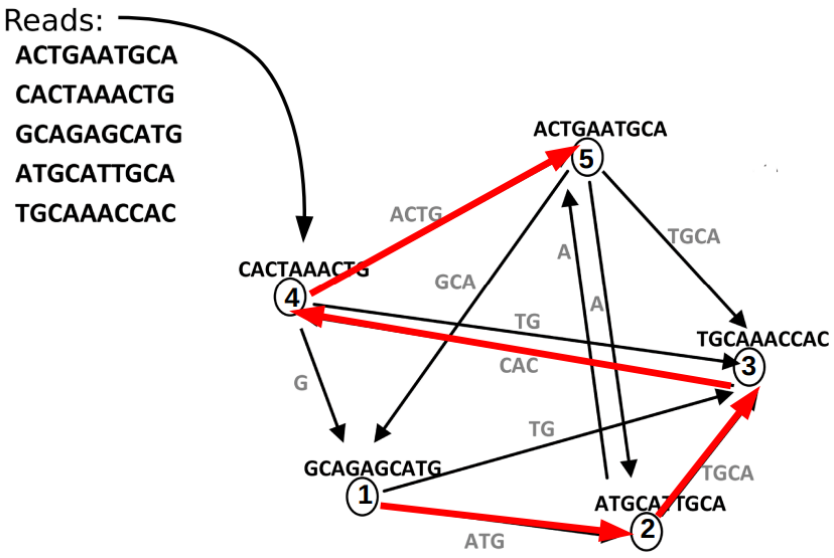
The Brujin graphs are more linear because we are only taking into consideration consecutive k-mers.



Assembly Graphs: Greedy

The greedy approach is based on local connectivity. So, it's a local approach (we are looking for the 3' 5' alignments).
The problem is that we have many connections and it will take the best alignment and will try to follow the best local alignments in order to catch the connections between all those reads and will produce a sequence.
This sequence, as it is based on local alignments, may contain misassemblies.

Maybe locally it's a good solution but it is not globally.



Assembling sequences with PHRAP

This is one of the first assemblers that uses the Greedy approach.

It uses long reads but still has the problem mentioned before.

It compares all reads against all, but pairwise and locally (source of errors in the final assembly).

We can have some improvements by trying to index the sequences. BLAST does this when we are doing an homology search. But at the end, it also does local alignments so, we can use k-mers to have an index (keywords) that allow us to capture which are the best possible pairs of homologous sequences.
These keywords are already calculated in the database and are calculated dynamically for the query.

So, an improvement of this approach is using k-mers in order to provide a set of candidate sequences that can align. Then it performs the Smith-Waterman alignment on candidate pairs.

- Find reads with matching k-mers
- Perform Smith-Waterman alignment on candidate pairs
- Create consensus sequence from high-scoring alignments

Assembly Graphs: OLC

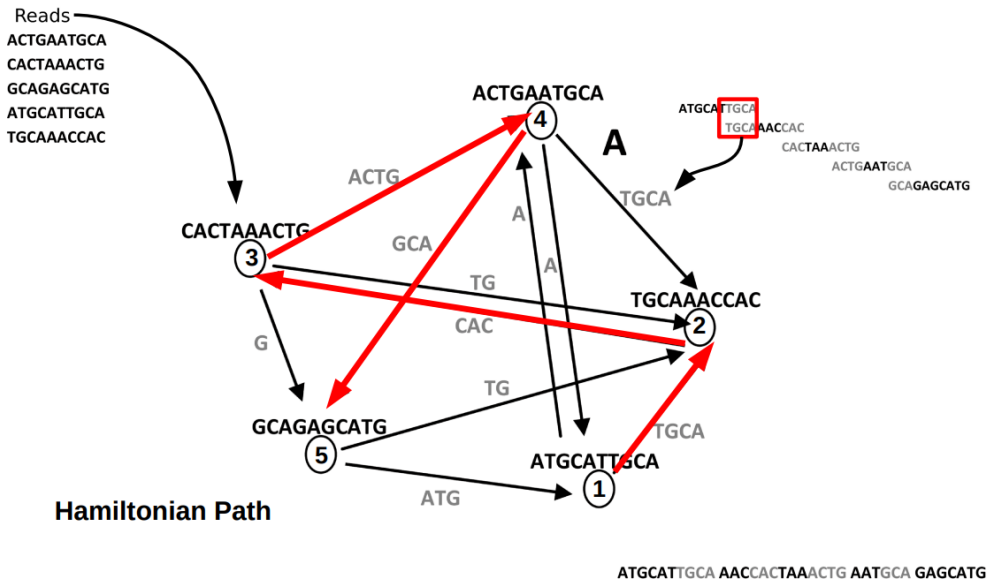
In the case of the OLC graphs we shift into global alignments.
So, first we try to get all the scores and then try to get, not the best local alignment, but the optimal set of alignments in the graph.

So, it's a global approach because it looks for the best assembly based on the best scores of the alignments.

The optimal path is the hamiltonian path, which tries to go through all the nodes at least once.

All the overlaps are translated into the edges and it looks for the best set of edges to start with the alignment.

With respect to the greedy approach, they have the same nodes and edges but the path is different.



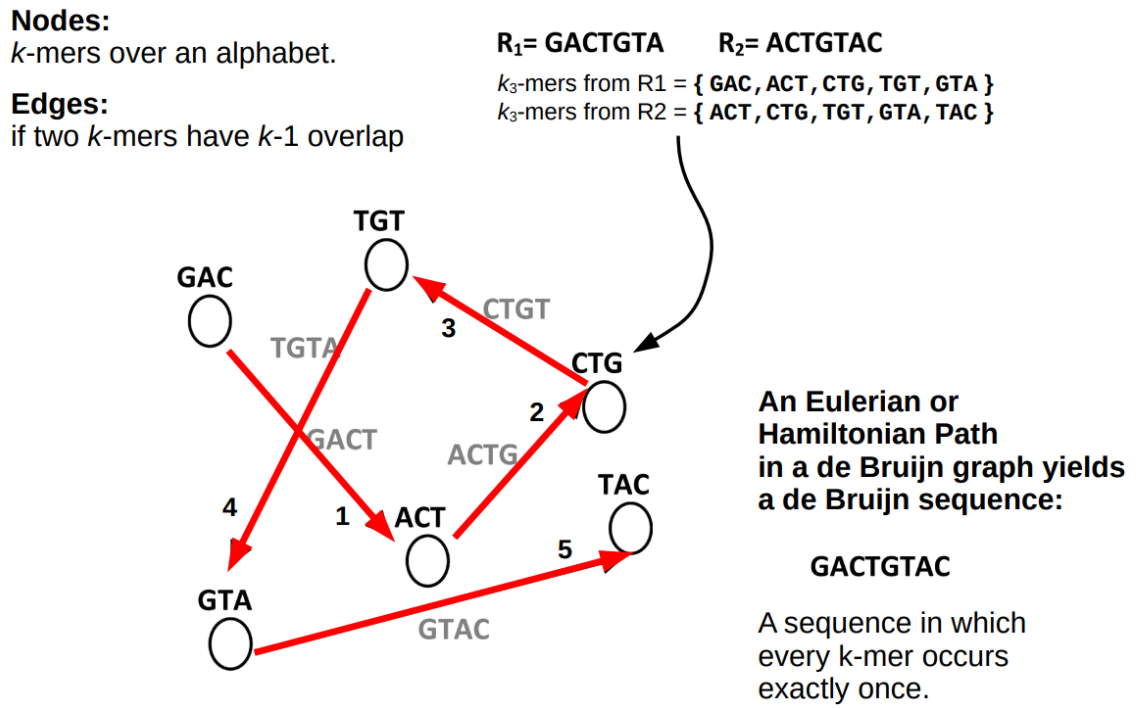
Assembly Graphs: de Bruijn

We have the nodes defined as k-mers and the edges defined as the k-1 overlap (window of 1). So, if we have an overlap of 4, there is a kmer of 3.

In this example we have the reads R1 and R2.
We define all possible k-mers of size 3 with a running window of 1. Then we project them into the graph.

In the edges we can represent the overlap or the composite sequence by overlapping the 2 consecutive k-mers.

Finally, we just find the optimal path through the graph.

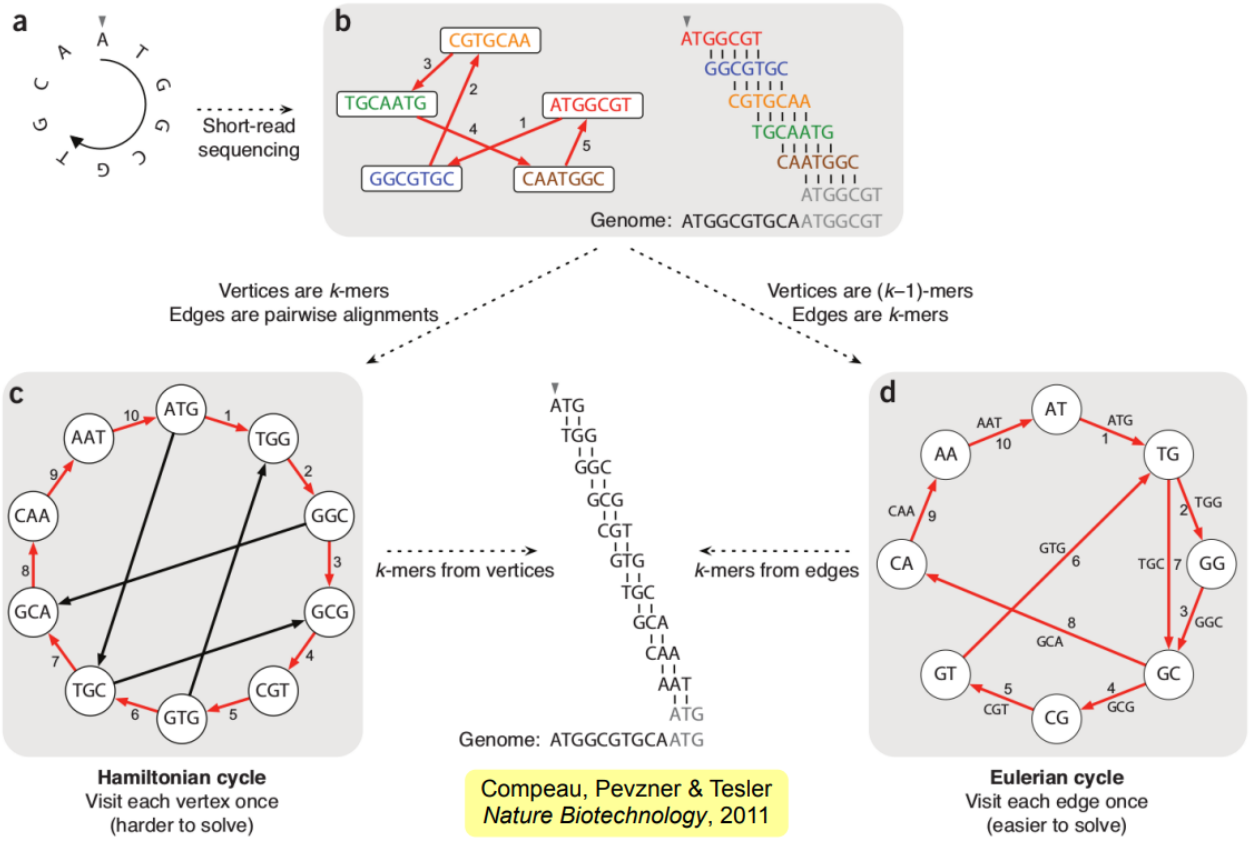


Note that the last graph was very simple but remember that there could be many possible paths going from one node to another.

Assembly Graphs

We have 2 possibilities to go through the graph:

- We have seen the Hamiltonian path, which visits once every single node
- There is also the Eulerian approach that looks for the path that is visiting edges only once. So, we can visit a node more than once but we can not visit an edge more than once.
  When we have repetitive sequences, this is a good approach.

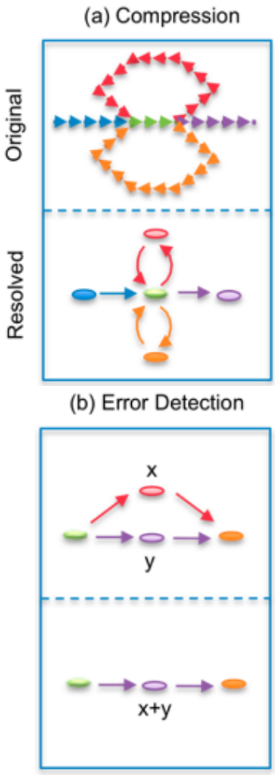


So, we can obtain the Bruijn sequence using an Eulerian or Hamiltonian path.
At the end, we obtain the same sequence but the Eulerian cycle is a little bit better, especially for repeats.

De Bruijn graph heuristics

When we are working with k-mer graphs, we can take into account some post-processing of the graph before looking for the optimal path:

- We can build a graph and directly look for the optimal path. So, we have hundreds of thousands of nodes and edges.
- Graph heuristics: We can try to simplify the graph before finding the optimal path.



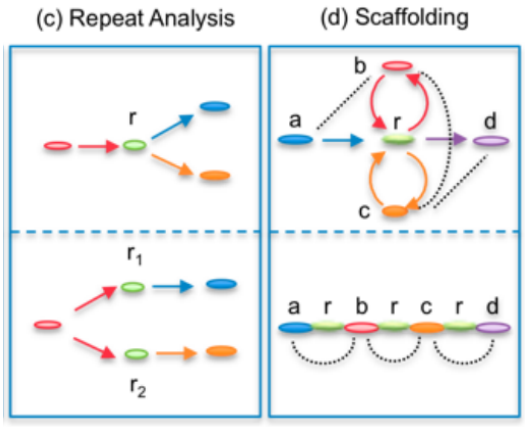
Compression: Getting less edges that our algorithm has to visit.
One example of graph heuristics is the String graph. Remember that it takes sets of consecutive linear paths of k-mers (which define one of the branches in the k-mer graph) and we can collapse them in a string.

So, this string is just the overlap of all those k-mers in the loop. So, we are just connecting fragments of sequences that have different k-mer composition and k-mers that are overlapping within that string.

Error Detection: We can use information from the reads to detect errors. Because as we are reading the reads in a FASTQ format, we have information about the quality score for every single nucleotide.
So, we can include this information to detect mismatches.
Note that just a difference in one nucleotide can produce a new branch in the graph (because we will have a different k-mer and hence a different overlap).

Repeat Analysis: We can do expansions

Scaffolding: We use this specially when we have regions that can produce loops. When our graph has a central node that has self loops, it can be a problem to know how many times our program has to go through that loop, which loop is the first one…
We can take information about the scaffolding. So, if we have information about the distance between the k-mers (obtained from the reads), we can project these distances into the graph and find the optimal path (then we will have no ambiguity that the loops produced before).



Once we have simplified the graphs, then we can look for the optimal path that allows us to reconstruct the genomic sequence.

k-mer size tradeoffs

We must choose the size of the k-mers we are going to use:

- **Large k:** Less ambiguity (the probability of finding the same k-mer by chance is very small), worse coverage. If we find a lot of repeated k-mers maybe we are in a repetitive region.
- **Small k:** More ambiguity (less unique k-mers), better coverage that improves the assembly.

We are not going to use a k-mer size of 3 as in the examples because every 64 nucleotides we will find the same k-mer by chance.

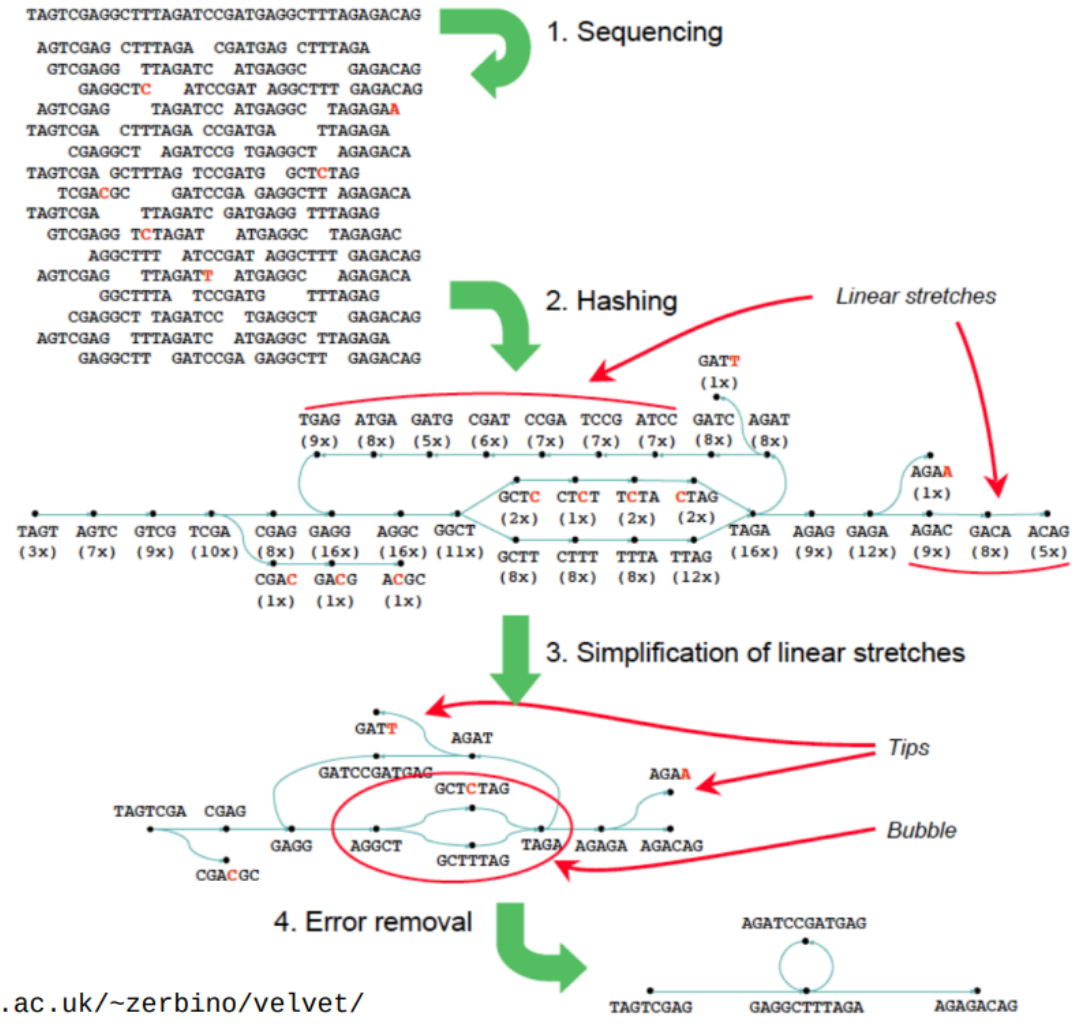
In practice k = 20 to 50
One strategy is to use multiple k-mer sizes and do many assemblies (then put in common all the assemblies that we have obtained and make a canonical assembly)

Tip removal & Bubble popping

Here we have an example with some real data.
We have a series of k-mers and we have made a Bruijn graph.

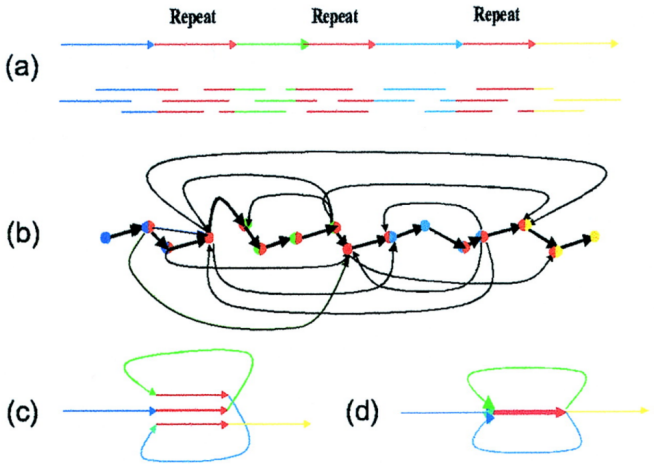
We can find “tips” in the graph, which will probably correspond to errors. Because as we can see in the coverage, the tips are k-mers that are repeated a low number of times (if they were alleles, then they would be repeated more times).
Thus, we must remove them.

Que passa amb les bubbles? Perque s’han d’eliminar?



Assembly Graphs: Repeats

In all graph approaches, the repeated will produce bubbles because we do not know which repeat goes first, second… So, it is really difficult to find a solution.



d) is the simplification when we apply all the heuristics to simplify the graph, but still, if we do not have scaffolds it will be difficult to find the correct path.

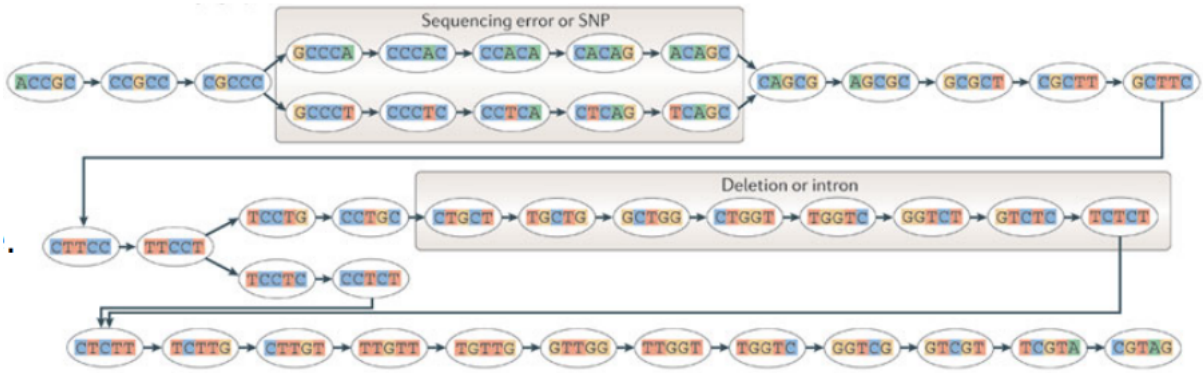
Due to the repetitive elements, we end up with chimeric contigs. Remember that we use the graph to obtain the contigs, which is the first step in the assembly.

Assembly Graph Errors

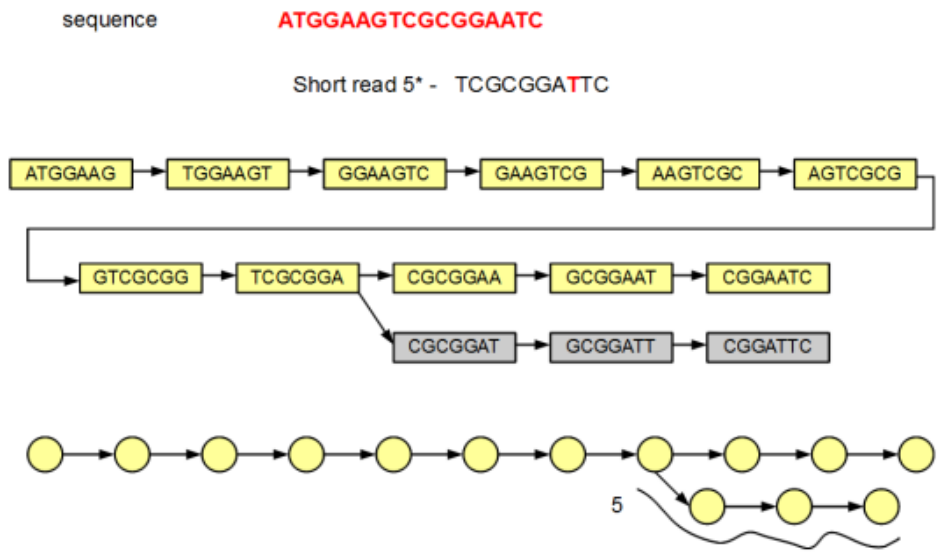
The typical errors we have in an assembly graph:

- **Read errors:** They correspond to the sequencing errors. We will have a bubble produced by the correct and incorrect k-mers.
  Complexity of the graph can explode.
  To solve this, we just look at the coverage.

  In diploid genomes, we also have different alleles (check the coverage). So, SNPs can not be removed, it can not be simplified (if we want to maintain the 2 alleles) and they will complicate the graph.
  Another problem are indels (don't know if it's an insertion or deletion), which produce bubbles that have branches of different lengths.

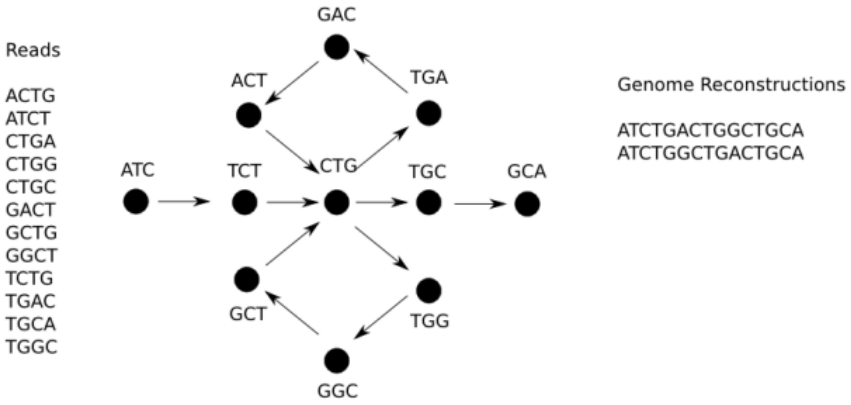


- **Filtering read errors:** In practice k-mers are counted. Low weight paths can be pruned



- **Eulerian path ambiguity:** Eulerian paths are not always unique! We have said that we can obtain graphs with a central node that corresponds to a repetitive region, which has some self loops that produce ambiguity.
  In this case CTG is a repetitive region.

  We normally assume that the Eulerian path is the best solution but in these cases it is not optimal.



Fixing Repeat Missassemblies

We have said that the programs can produce assembly errors.

But can we detect how well the assembly was done? If we know the initial sequence, we can assess the accuracy of the program.
But if we are annotating a new genome, we don't know how well we have performed our assembly.
So, once we have done the assembly, we can map the reads back to the genome. We would do that anyway because we want to know, for instance, the coverage through the genome.
So, we need to find the coverage to find which are the regions that are better supported.

When we are mapping the reads over the assembled genome, we can focus on the expected distances of the reads. I know that the reads, if I am using Illumina of 100 nucleotides and the expected size of the insert has 400 nucleotides. Then the distance between 2 consecutive paired reads must be 200 nucleotides.

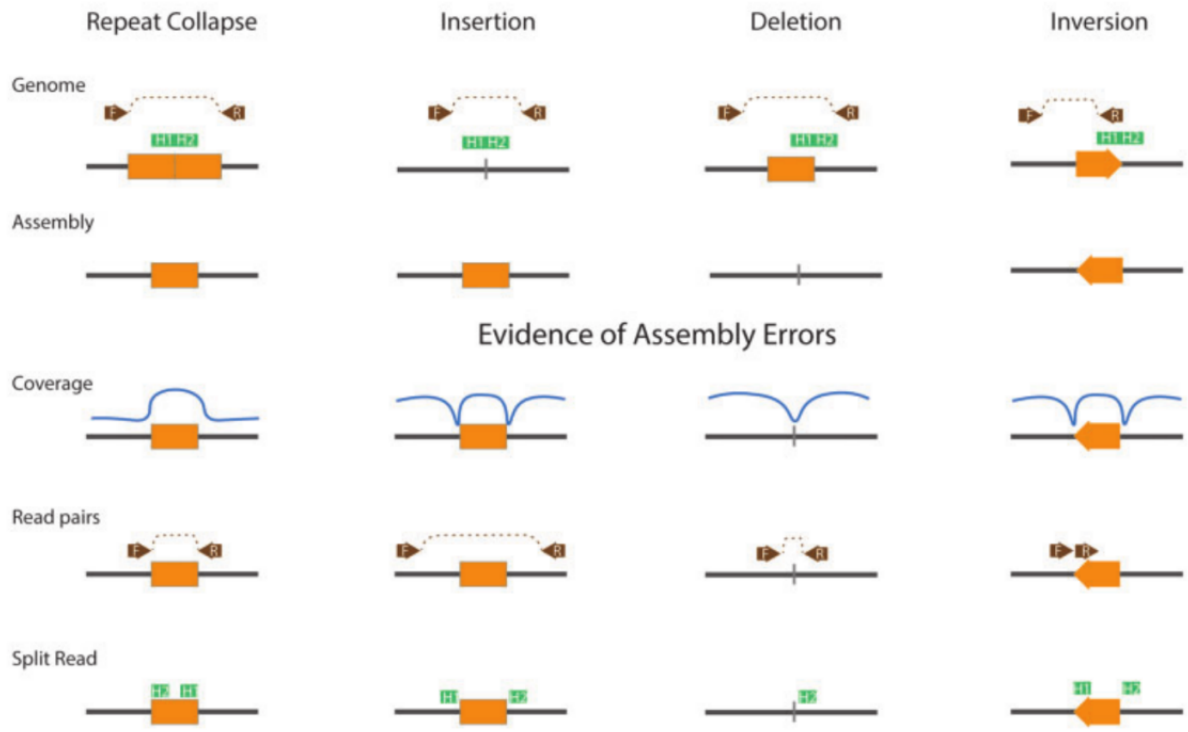
So, when I map those reads to the genome, those 2 reads must be at 200 nucleotides.
So, once I have mapped all the reads, I can look at the distances between the paired reads and assess if they are at the expected distance.

Type of Assembly errors

There are other types of errors that can be detected using the alignment of the reads over the genome.

If we have a repeat collapse, we will find that it has double the coverage.
We can use the same approach to detect other errors:

- For insertions, we will have many reads that are far apart from what we expect and we will have a drop in the coverage in the regions before and after the insertion.



Estimating Genome Size

There is some experimental data about the length of the genomes:

- They have the molecular weight of the DNA of different species, which is computed by simply doing a gradient.
- We can also use a known genome and compute its length or weight (we know the weight of each nucleotide)

But for a new genome, we do not know anything and it would be valuable information to estimate how well assembled a genome is.

The k-mer counting methods can be used also for estimating genome size using the depth and length of reads and the depth (how many times a k-mer appears) and size of the k-mer.

$$D = \frac{D' \cdot l}{l - k + 1}$$

and

$$G = \frac{N_{base}}{D} = \frac{N_{read} \cdot (l - k + 1)}{D'}$$

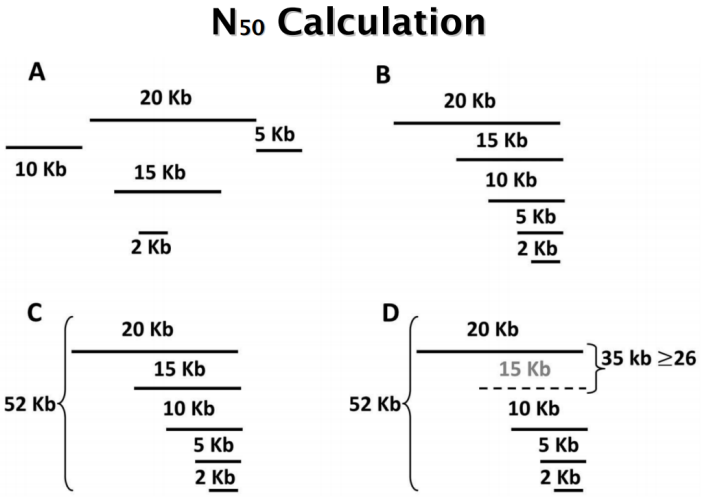
where:

D ~ estimated depth of reads;
G ~ genome size;
l ~ average read length;
k ~ k-mer size;
D' ~ k-mer depth at the peak of the k-mer histogram;
$N_{k\text{-mer}} = (l - k + 1)$ ~ number of k-mers in a read;
N_{base} ~ number of sequenced bases,
where $N_{read} = N_{base} / l$ ~ number of reads.

This method is useful because genome size can be estimated before or without whole-genome de novo assembly.

If we have the genome size, even if it is an estimate, we can somehow evaluate the quality of the assembly. Because we can have some calculations like the N50 or the N90…

So, we can calculate how many contigs we need to cover a % of the genome.
N50: How many contigs do we need to concatenate to cover 50% of the genome.
We use the biggest contigs until we reach more than 50%.



The smaller the N50, the better!

We can also compute **L50**, which corresponds to the length of the last contig I am adding in order to arrive at the 50% of the length. In this case it would be 15kb.

The larger the L50, the better!

RNA-seq Libraries

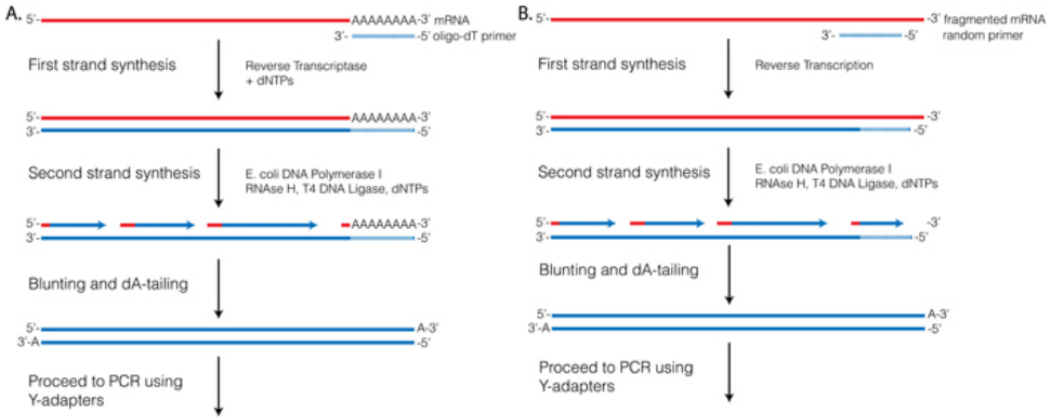
First of all, remember that we are sequencing DNA and, hence, we have to retrotranscribe the RNA into cDNA.
Depending on the method, we can get more or less coverage of the fragments.

In this case, we use a primer that is complementary to the poly-A tail. So that we are targeting the mRNA.

- If we have a tRNA, we will not have this tail and we will use random primers that will hybridate to the specific RNA.

Once we have the cDNA, we use random primers to obtain the initial sequence.

Note that in RNA we have one strand and in DNA we have 2, so we will need to identify which is the coding strand (F or R) that defines the mRNA sequence.

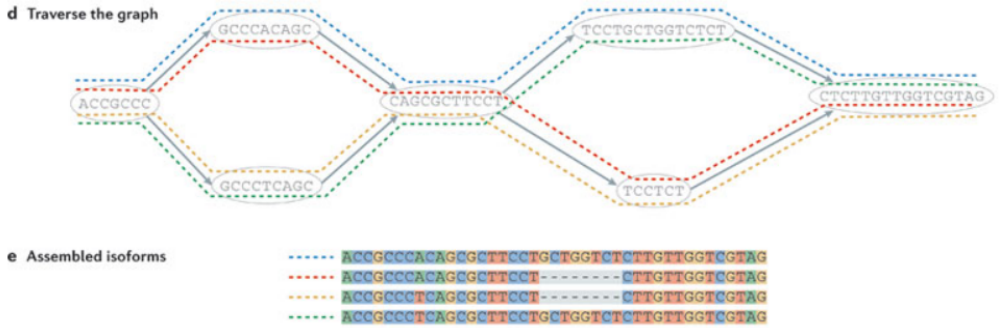


In complex organisms, we have an extra problem because we have alternative splicing.
If we work with mRNAs, we won't have introns (hence we will have less repetitive elements than a genomic assembly) but the constitutive exons work as a repetitive element.
Because if we have several transcripts, all of them will share the constitutive exons.
Thus, it will look like a repetitive element.
So, multiple isoforms can inflate ambiguity in the Bruijn graph.

So, we have problems in the assembly because we have shorter sequences, we don't want to get the contiguous sequence of all the mRNAs connected, we want to recover each individual mRNA.
Also, if the mRNA has different splicing isoforms, we can end up with complex subgraphs.

Transcriptome Assembly: Trinity

If we apply the same algorithms as we use with genomic assemblies, we can end up connecting all of the transcript reads into a single contig and we do not want to do that.

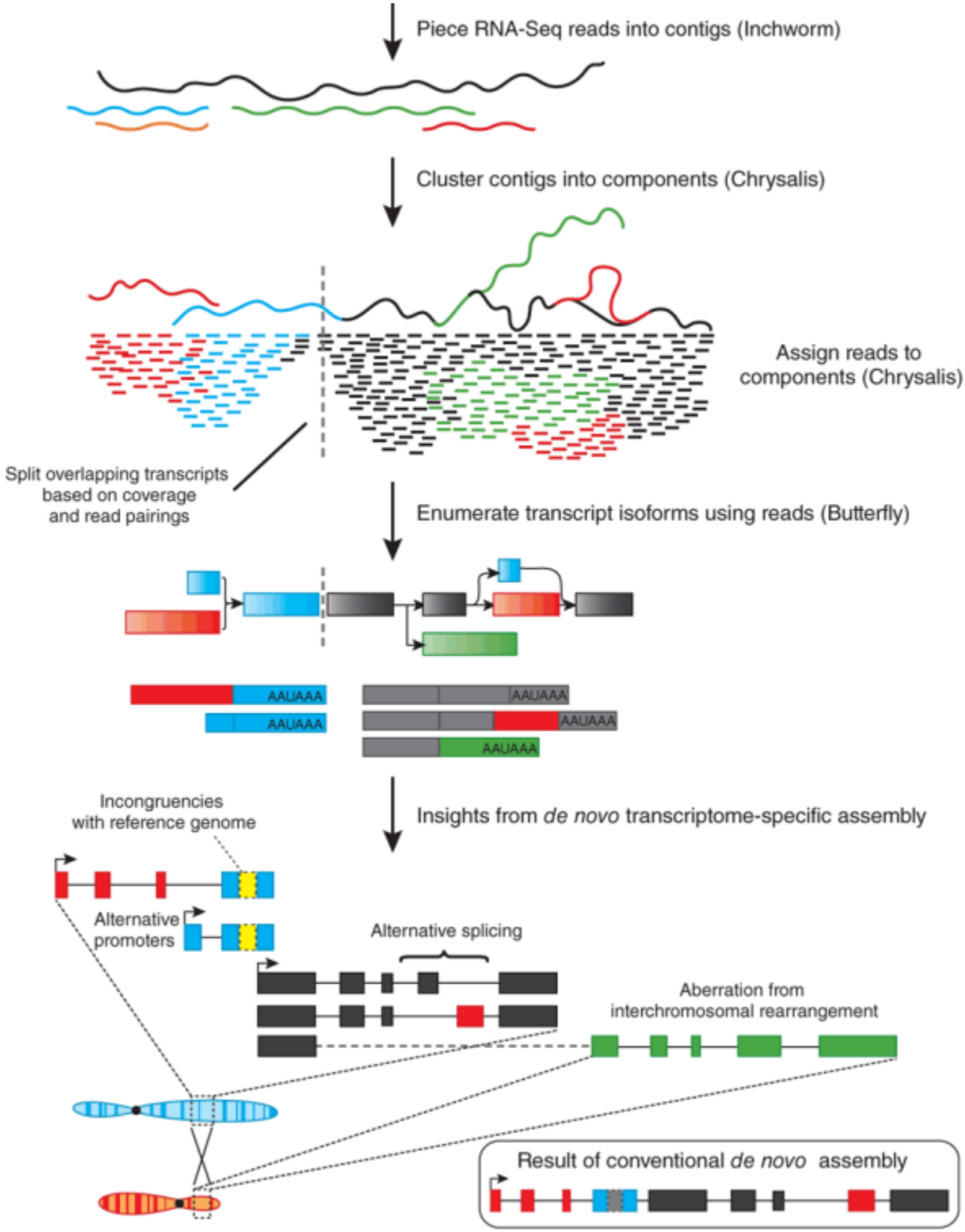


So, specialized tools like Trinity only assemble RNA-Seq data because they take into account that we want to get the individual mRNA.

In an initial step, we try to get the longest contigs from the assembly graph (Chrysalis). Then, it maps back all the reads and in function of the coverage, it detects breaking points. The big changes in coverage define different transcripts.

Moreover, the differences in coverage are used to quantify the gene expression.

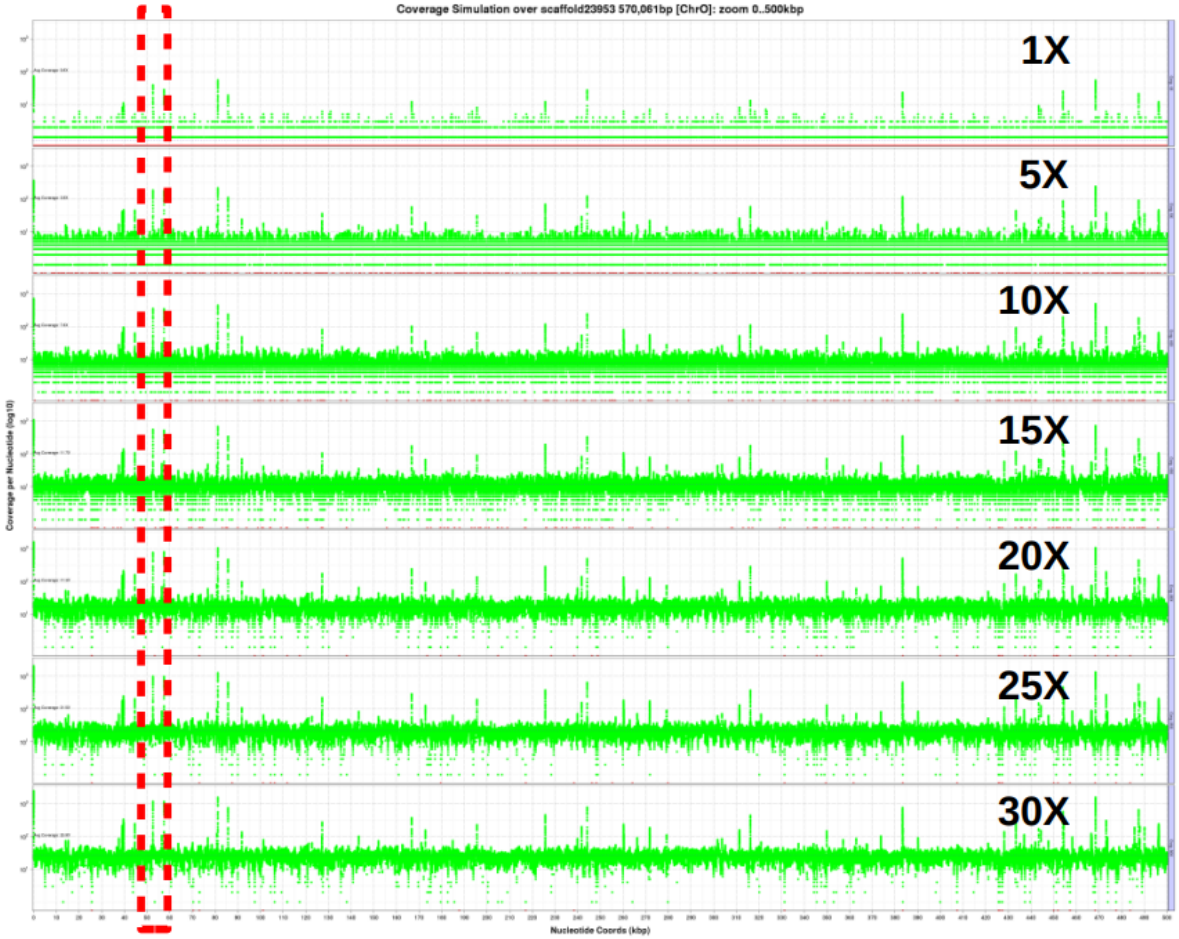
So, depending on the biological material we want to assemble (DNA or RNA), we need to take into account some properties.


Piece RNA-Seq reads into contigs (Inchworm)
Cluster contigs into components (Chrysalis)
Assign reads to components (Chrysalis)
Split overlapping transcripts based on coverage and read pairings
Enumerate transcript isoforms using reads (Butterfly)
AAUAAA
AAUAAA
AAUAAA
AAUAAA
AAUAAA
Insights from de novo transcriptome-specific assembly
Incongruencies with reference genome
Alternative promoters
Alternative splicing
Aberration from interchromosomal rearrangement
Result of conventional de novo assembly

Assembly coverage

The peaks are repetitive elements (reads that map more than once because they have repetitive elements).
The valleys are sequencing gaps. So, they are regions where we do not have reads. If they appear in high X, it means that it's a region that has problems to be sequenced (there is a GC bias or something).

They are both in all X.



The sequencing depth is important in order to be able to reconstruct and to determine if there are errors or SNPs in an assembly.
But the more sequencing depth, the more expensive the analysis.

7th Theoretical Class

Indexing Sequences: Fast Mapping

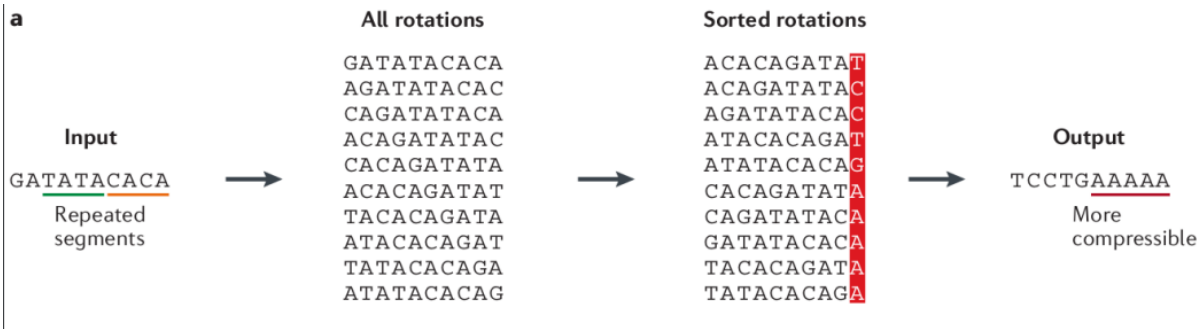
As the amount of sequences we produce is growing exponentially, and we have billions of sequencing reads that have to be mapped back to the assembled genomes or transcriptomes to find the coverage or to get those reads aligned because we want to discover new variants…

All the analysis we can perform later on on those assembled sequences or transcriptomes depend on mapping, which is the general case of aligning sequences (remember that when we talk about alignment we strictly speak about Smith-Waterman or Needleman-Wunsch dynamic aligning approaches, in which we use models for substitution and gaps).

We also said that we can use indexed words in order to speed up the searches, like BLAST (heuristic approach).
We can take that a little bit further and use modern aligners. They map exact words between the set of reads we want to align against the assembled genome or transcriptome.

The fast mapping depends on k-mer indexing. So, we use fixed size k-mers that will be the words defining the indexes of the reference sequences and for the reads to be aligned.
Here we have 2 approaches to start indexing the k-mers:

- For a given index, we can generate a new index that is more suitable for fast searches in the database. So, from the raw set of sequences (either from the database and the query), we get lists of indexed kmers that we will use to search for matches between the sequences.
    - It takes a while to make rotations of the original k-mer
    - Sort them
    - Obtain the last nucleotide of each k-mer and join them forming a new index that will be faster and that will be more compressed



Remember that assembly algorithms take all the k-mers from the reads and use them in order to build the assembly graph and find the optimal path. But the information about the reads is somehow lost. So, once we have the assembly, if we want to know where the reads are located in the assembly, we can just map them into the assembly.

So, there are 2 steps. Decompose in k-mers and then go back to the reads and map into the assembly.

- The other approach is not explained.

Mapping Algorithms

Sometimes we combine several of these approaches in order to get a better performance when we are working genome wide or transcriptome wide (work with billions of sequences to map).

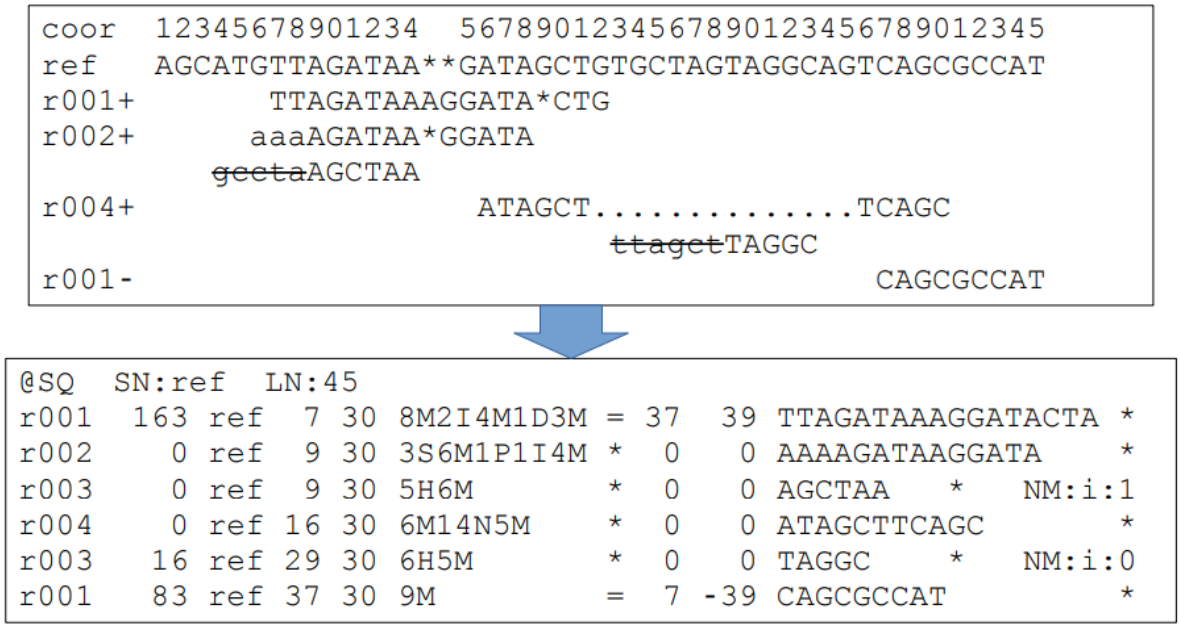
The main disadvantage is that they are looking for exact or almost exact words and, hence, they only allow for one or two mismatches. Also, gaps are really difficult to model.
But they allow us to map in a very short time.

SAM/BAM files

The standard alignment visualization requires a lot of space and, hence, they created the SAM/BAM format to store the information about the alignments of all the reads:

- We have a single line per read. It starts with the read code.

This alignment is translated into the “Siga Alignment String” in which we have the alignment encoded with characters. So it is a way to compress all the information of the alignment.



Analyzing RNA-seq Data: an Overview

We saw that if we have a higher coverage we will obtain a better assembly (we will have less gaps and we will have larger contigs). If we are analyzing RNA-Seq data, there are many things that we can take into account.
Here we have a summary of the analysis we can perform in the RNA-Seq experiment:

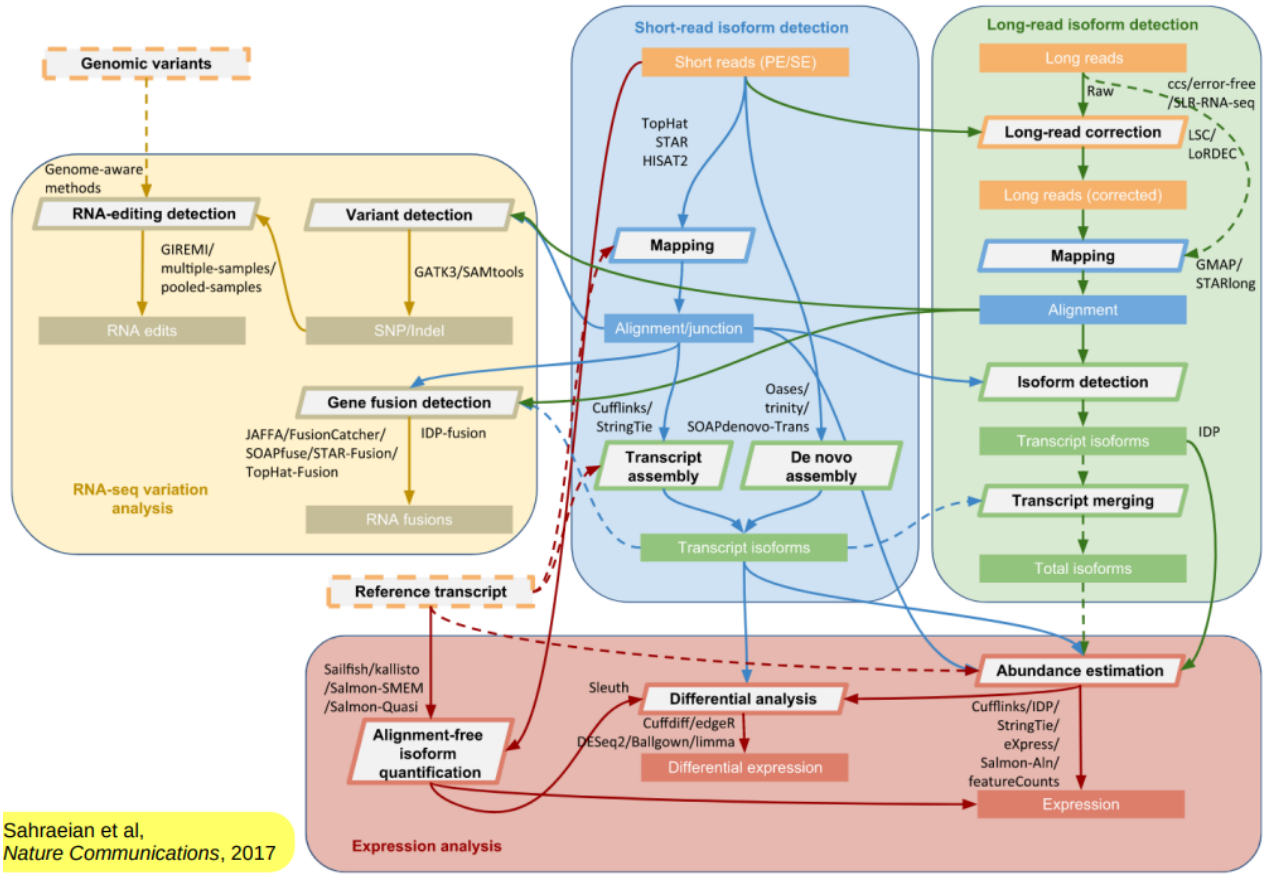
- Detection of variants
- Quantification of expression levels

We can start with long (nanopore or PacBio, for example) or short reads (Illumina).
At the end, we are going to recover a reconstruction of the transcriptome (set of splicing isoforms that are being expressed in a given experimental condition) and we can use or not a reference genome to find the regions that are being expressed.

Remember that we want to avoid reconstructing chimeric transcripts that can combine several transcripts into one.
The idea of the genome assembly is to obtain the optimal longest path that defines a chromosome. But in the case of the transcripts, when we are in the initial step of “Trinity”, we had the danger of combining several transcripts into a longer chimeric contig because the initial assembly step looks for the longest path.

Remember that Trinity looked for changes in coverage to split and try to recover individual transcripts.
So, in de novo assembly, we always have issues in order to reconstruct unless we are using long read isoform sequencing in which probably one of the reads contains the whole transcript.

Moreover, remember that we have an additional problem with constitutive exons, which act like repetitive regions.



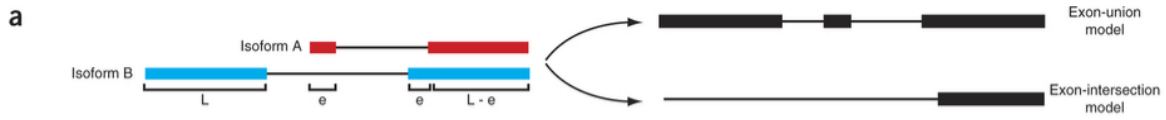
Sahraeian et al,
Nature Communications, 2017

RNA-seq: Isoform Quantification

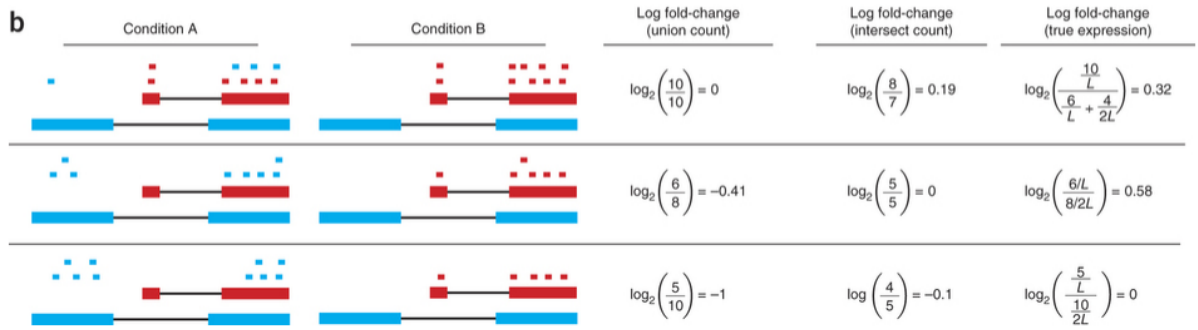
Once we have the splicing isoforms described, then we can compare the expression levels. The expression levels depend on the coverage. So, once we have a list of putative splicing isoforms, for example, in this case:

- Here we have 2 splicing isoforms that have a constitutive exon (more or less) and 2 alternative exons. So, the final product will be different.

So, once we have the complete catalog of splicing isoforms, when we map the reads back to the assembly, we can find the amount of reads supporting each of the exons and, hence, quantify the expression level.



We can compare 2 conditions by sequencing the same transcriptome under different experimental conditions.
Thus, we will have different sequences and different coverages in different conditions.



How do we transform the coverage into translation levels?
We just count how many reads fall inside the exon. We can use different measures or ways to calculate the changes in coverage.
At the end, we transform the mapping into a translation table in which we have the number of counts, RPTM…

Finishing Genomes: Physical Maps/Markers

We used sequencing to assemble a genome or a transcriptome.
We can resequence, but at one point, we can jump from the 3rd step to the 4rth in the assembly.

Remember that there are 4 steps in a assembly:

- Build contigs
- Scaffolding (detect the splicing isoforms for transcriptomes)
- Closing gaps
- Finish the annotation

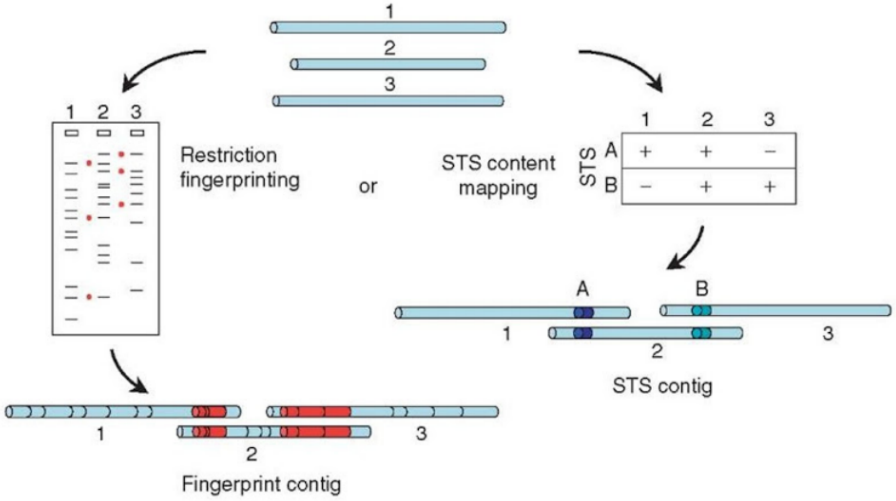
At genome level, if we are working with scaffolds, we can still try to place the scaffolds over the chromosomal sequences. Experimentally, we can perform some analysis that allows us to locate regions in chromosomes.

When cells are dividing, we have the chromosomes compacted and we can perform the FISH experiment that consists in mapping a probe that has been marked with a fluorochrome.
We know the sequence of that probe and we just hybridize that over the chromosomes when the cells are dividing.

If we have different probes labeled with different colors, we can localize them with the FISH technique. So, we can look at which physical position of the chromosome the probe binds. So, we can use this as physical markers.

Another physical marker would be a known sequence that is recognized by a restriction enzyme. So, the enzyme breaks the chromosome and we can create a physical map.

If the target sequence of the restriction enzyme aligns with our scaffolds, we can just locate these scaffolds using these markers.



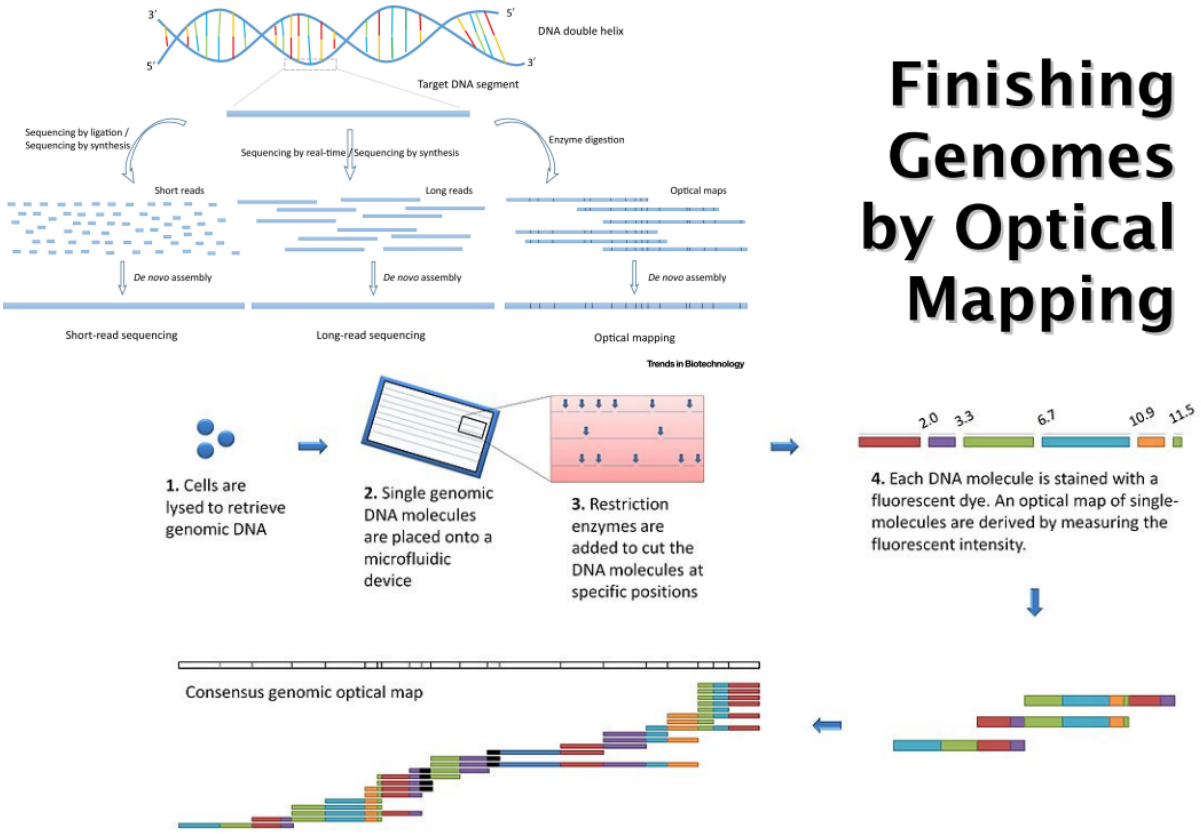
Finishing Genomes by Optical Mapping

We have different ways to obtain the assembly:

- Short reads
- Long reads, which get a better assembly of the scaffolds

We can also use enzymes that will break the genome and thus we can generate a series of optical maps with a variety of marked probes. Instead of doing a FISH over the chromosomes, we can just put the molecules over a slide/glass, obtain the linearized molecules and hybridize many probes at the same time.

We will obtain a colored barcode of the molecules (different molecules will have different barcodes) and what we are going to align are the barcodes instead of the sequences.



What happens with the sequences that are not supported by scaffolds in any of the mentioned approaches (restriction enzymes, genetic map with recombination distances, optical map…)? We just add NNNNNN

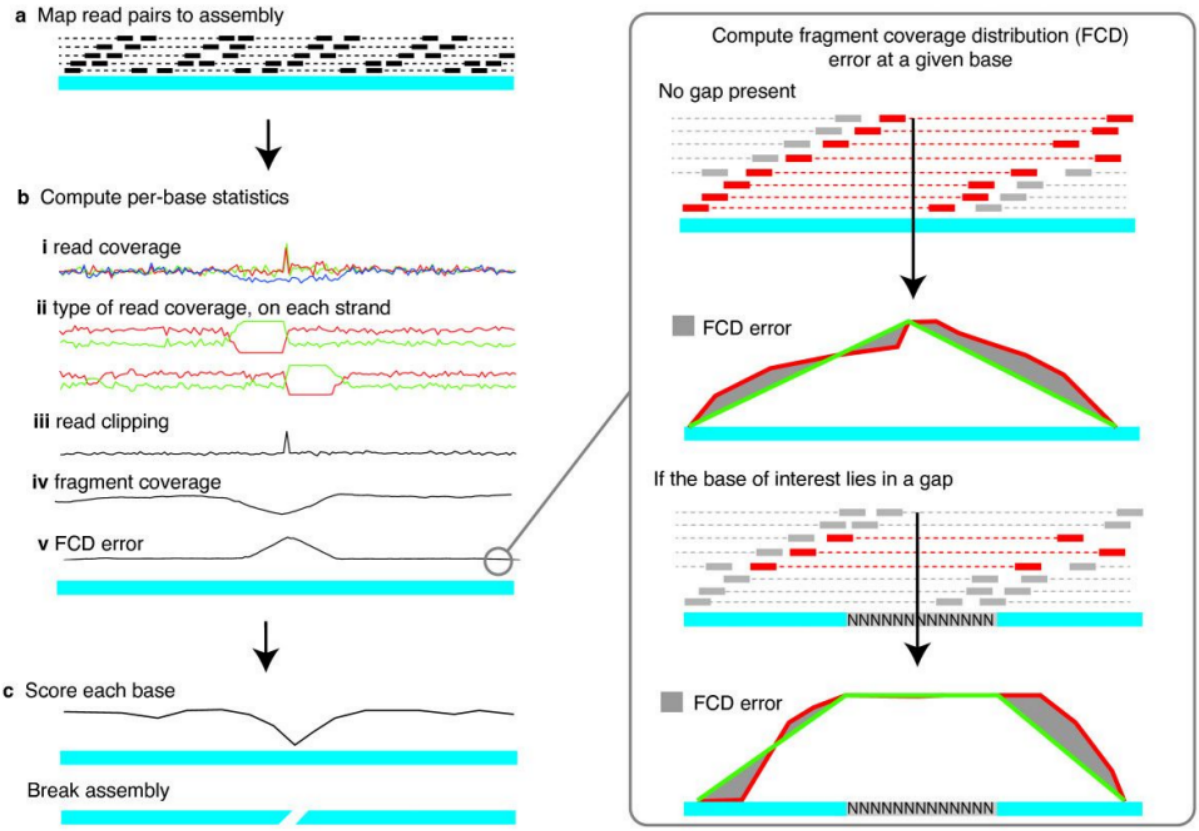
Another approach would be using the High-Seq technologies. We know that the chromosomes are folded in the nucleus and thus they are not randomly distributed. They always have a specific conformation and we can use the High-Seq technologies to break the DNA that is not bound to structural proteins and find contacts at genomic scale.
So, we can find short and long distant contacts or even interchromosomal contacts.

REAPR Assembly Correction

We can also check the assemblies (apart from performing this finishing step) using REAPR to check automatically the coverage. We can also use a viewer to look for regions that may look as misassembled or that have a huge variation in the coverage. But REAPR will take the alignments from the reads mapped against the assembled genome or reference genome and can calculate different statistics.
Different misassembly errors (indels, inversions…) had different patterns in coverage.

So, REAPR can process all the coverage and will try to find the regions that can be chimeric contig. It will split the contigs into 2.

So, we are breaking the contigs and, hence, eliminating the possible chimeras.



Assemblathon [1/2/3]

We have different projects that give us metrics to assess how well we have assembled a genome using different programs.
We just compare the programs by looking at the N50, L50…

They also provide different formulas from the alignment of the assembly with respect to the reference genome. So, we have the contigs assembled. We align back the assembly to the reference genome and calculate other metrics.

The idea was to discover which was the best program.

Assembly Assessment: GAGE

Independent team that not only performed the assessment but also provided the protocols to generate assemblies that were then evaluated:

- If they were going to evaluate SOAPdenovo, they provided the protocol they used to produce the scaffold with that tool.

Accuracy Assessment: Conserved Genes

How well we have assembled the genome depends if we have connected the reads properly (we have tools based on coverage that check this). So, until now we have only been focused on linearity.

But what about completeness? Because we may still have a lot of gaps.
These are annotation tests:

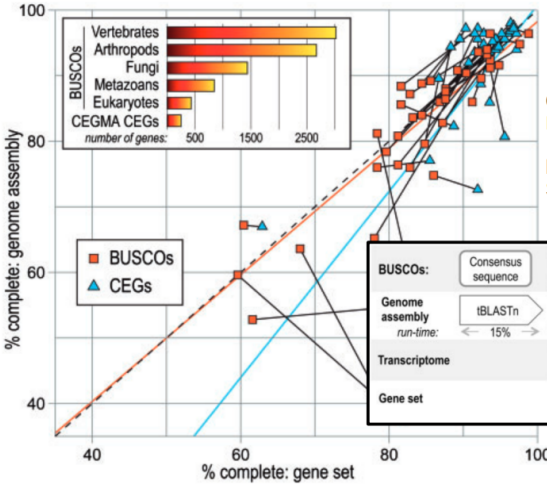
- CEGMA and BUSCO tools contain a set of ultraconserved orthologous genes that appear once on most of the species.
  So it's a gene that is orthologous in all the species and is highly conserved (housekeeping gene). Thus, it is easy to find in the genome.

  They just get a catalog of those genes that must be in all the species. So, if the assembly is good, we must find these genes.

  BUSCO also classifies the ortholog genes by taxonomic group, so we can specifically look for ultraconserved genes in a certain taxonomic group. The more close to the taxonomic group of the genome you want to assess, the better.
  The more complex the group, the more BUSCOs you will have to check.
  - There are a few genes that are shared across all eukaryotes but there are many more genes shared within vertebrates.

The idea here is that we know those genes must be in my assembled genome.
I map them over the genome and quantify how many of them are found, how many are not found, how many are partial…

So, when we are assessing the quality of an assembly, we can do this protocol.
This can also be applied to validate transcriptomes.



As we can see in the graph, there is a correlation between the completeness of the genome assembly and the completeness of the gene set (number of genes that have been found at the BUSCO set).

So, it is important to check the contiguity but also the completeness of the assembly.

Metagenomic Approaches

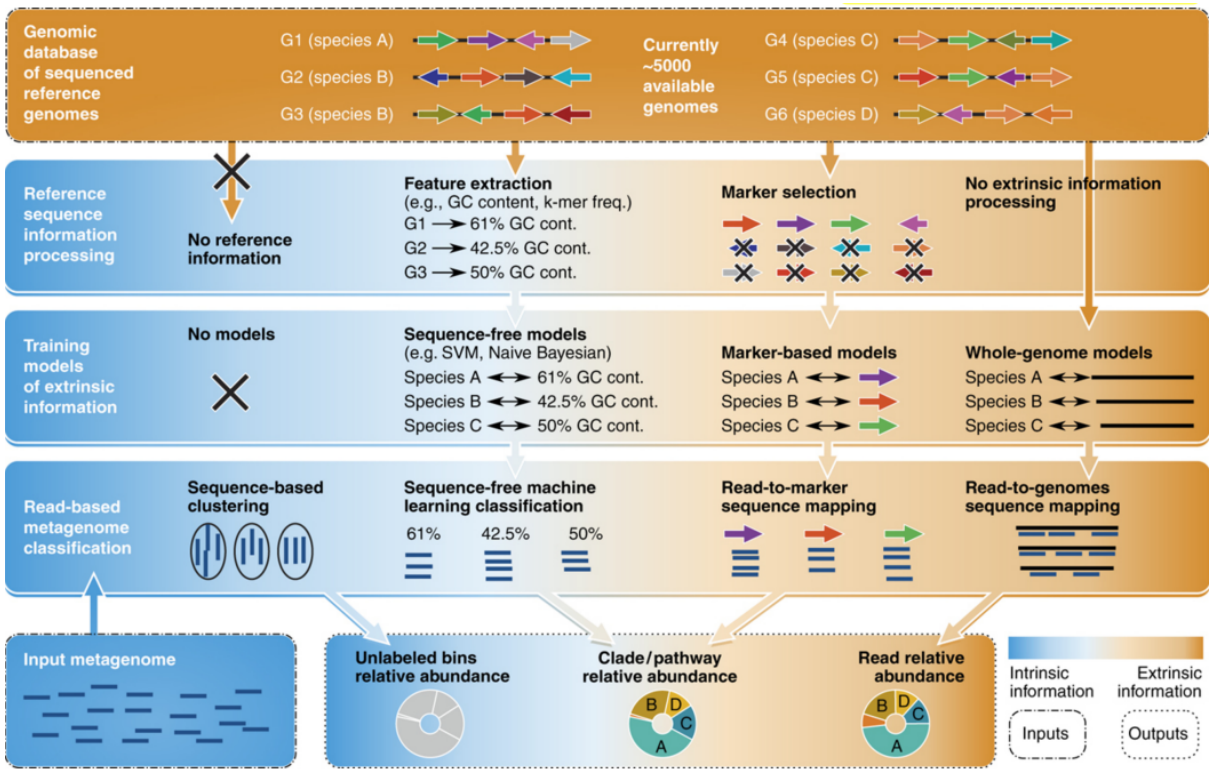
What happens with metagenomic samples? It's a problem similar to the one we have with transcriptome assembly, in which we have a mixture of genomes.

This table also summarizes many of the approaches we can apply in a metagenomic analysis.

Here the focus is not only reconstructing the assembly but also to assign a group and quantify how many times this group appears. So, I have a mixture of 5 species, so the idea is to:

- Reconstruct the 5 genomes
- Find how many times each species appears in the sample. Here the coverage will be an indicator of the abundance of a given species in the sample.

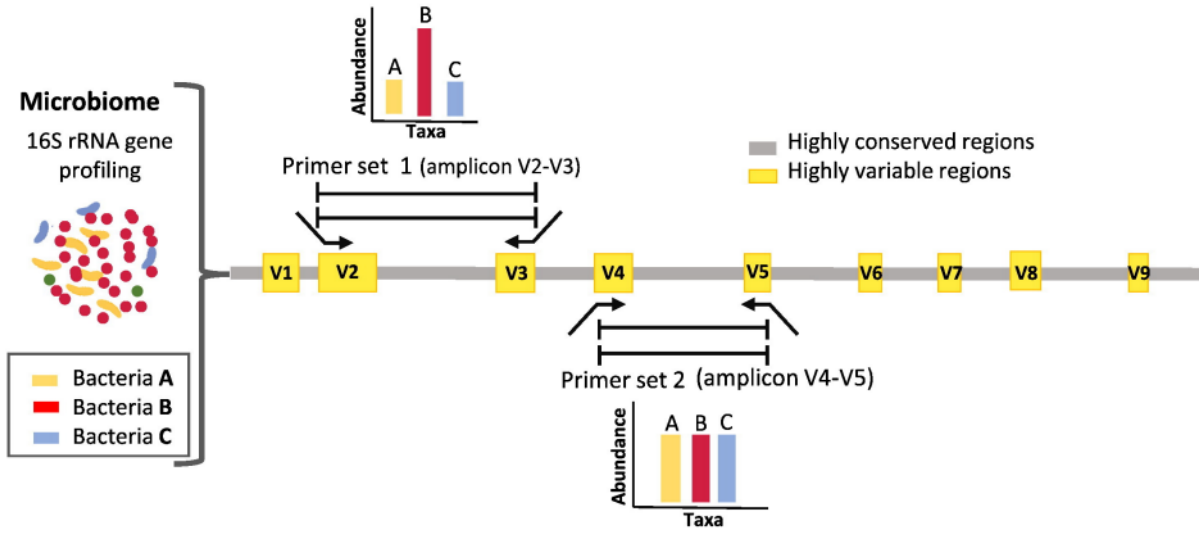
In brown we have extrinsic information (things that we already know), like known genomes or things that are derived from what is known.
So, we can use those general features derived from known genomes in order to classify the reads we have in the sample.
We can also use the markers for a known genome. If we find the markers in the sample that have a match for a marker that comes from a known species, we know for sure that that species is present in the sample.



Prokaryotic 16S rRNA libraries

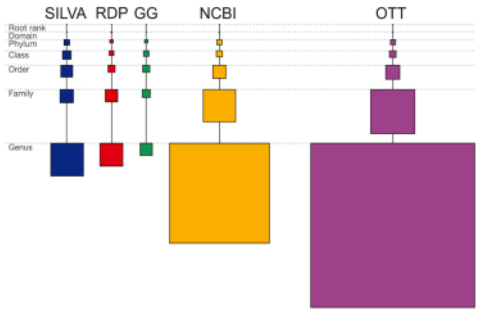
Here we have an example of usage of markers. One of the most typical markers is the ribosomal RNAs (16S for bacteria and 18S for eukaryotic protozoa).

We have special primers that are specific for variant regions of a given ultraconserved gene. The primers amplify a series of variable regions taking advantage of the conserved regions that are surrounding the variable regions.
So, when we sequence, we have the conserved regions that allow us to classify within a group and the variant regions allow us to perform a phylogenetic analysis in order to refine the group (find novel species).



Taxonomy	Type	No. of nodes	Lowest rank
SILVA	Manual	12,117	Genus
RDP	Semi	6,128	Genus
Greengenes	Automatic	3,093	Species
NCBI	Manual	1,522,150	Species
OTT	Automatic	2,627,066	Species

Balvočiūtė & Huson, *BMC Genomics*, 2017



This is amplicon sequencing, this is not genome based metagenomics.
So it's targeted sequencing. We have primers that amplify only a region.

There are different databases that can help us to assign or to classify the species in the sample. So, we can quantify the abundance of each species.

The problem is that the databases are not proportional in the number of species or the resolution they have at different taxonomic levels.

Note that these DBs do not store the whole genome but the 16S RNA. So it is faster.

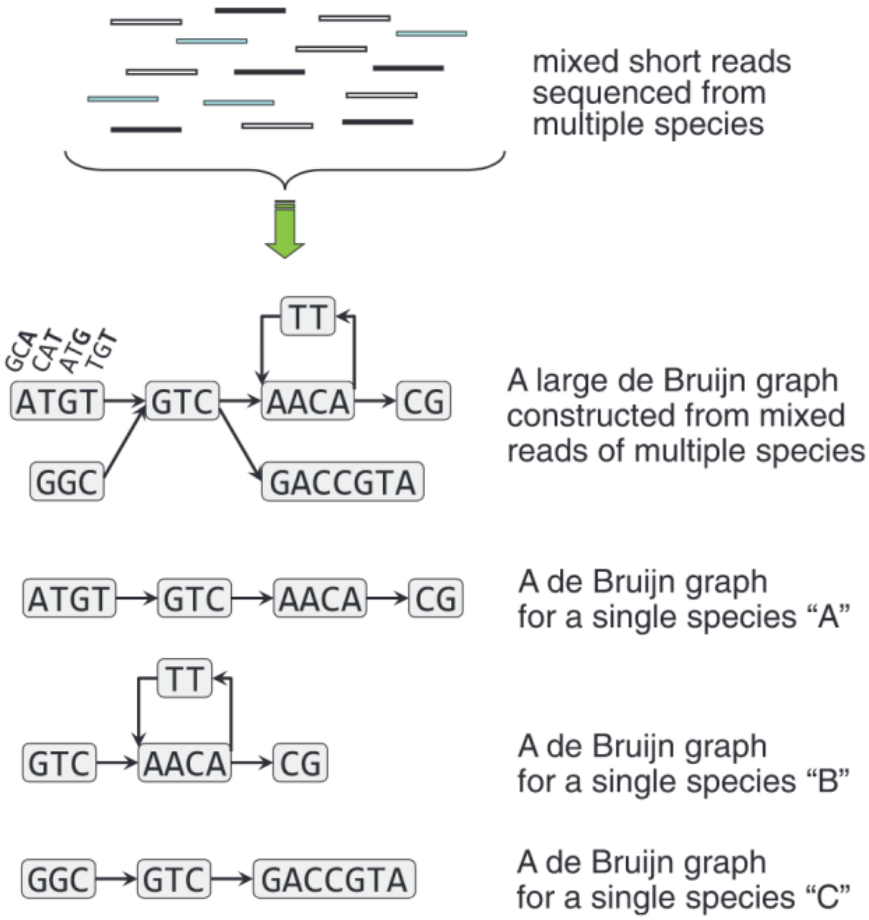
Metagenomics Assembly: Sample Mixtures

Similarly to what happens in the transcriptome assembly, when we are assembling a sample from metagenomic experiments we want to classify each fragment with respect to their species. But a priori we do not know to which species each fragment belongs.

We have to process the Bruijn/string graph slightly differently.
The tips in the graph define different species. So, at the end, once we have performed the graph we need to extract the sequence for each of the species.

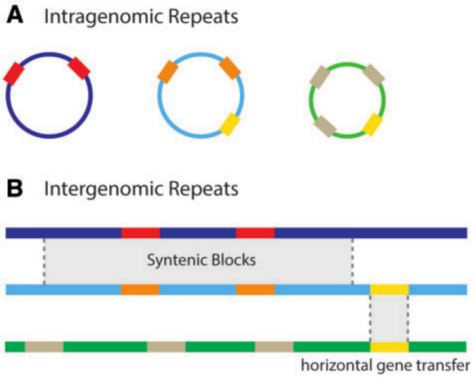
- Each subgraph will correspond to one species genomic sequence

This is a combinatorial problem, because if we have a lot of tips, it is going to be very difficult to separate the species.



Repeats in Metagenomes

We can have families of repeats that appear in the same species or the same taxonomic group. So, here we have a repetitive region that is shared because those 2 genomes are close



The repetitive elements can make the assembly difficult because they can collapse. They have similar GC content and thus they will collapse.

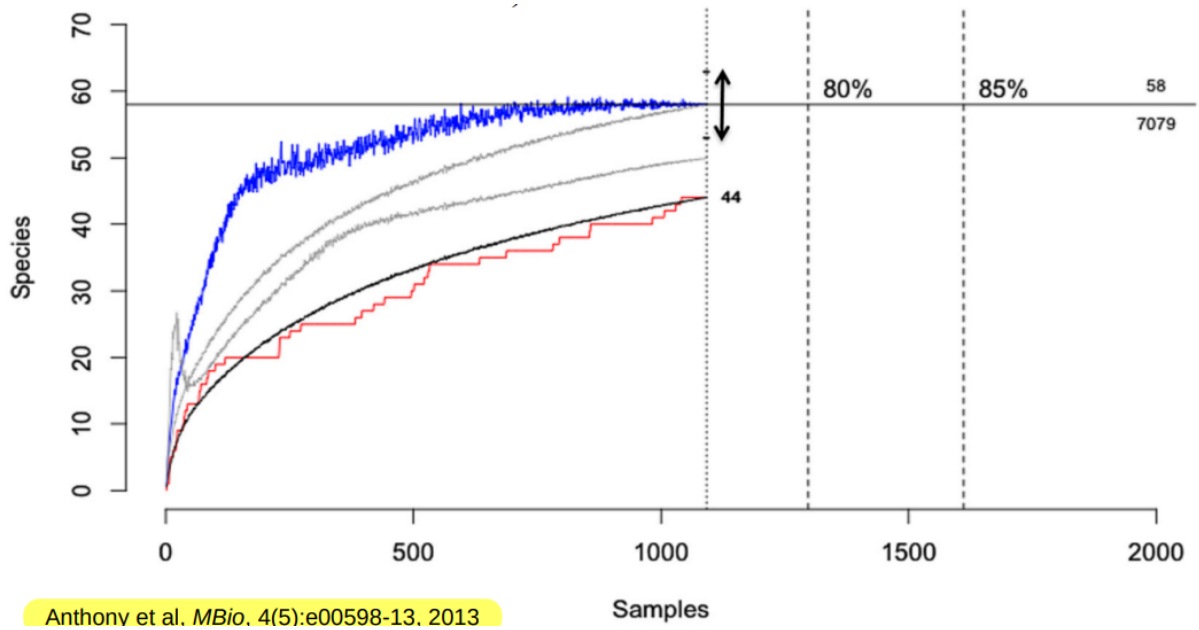
In bacteria, we have horizontal gene transfers. This is also going to be a problem. Because they look like another species.
If we have the 3 species, the HGT will also look like repetitive regions (similar to constitutive exons).

Estimating “virodiversity”

We always refer to known sequences to assess or validate…
We have an issue with the completeness of the DB.

If we are looking for what is in the metagenomic sample, the assignment of the species also depends on the completeness of the DB. So, we need to be aware of how complete the databases are.

In the case of the viruses, for instance, we have an analysis in which they were looking at the minimum amount of samples they need to process in order to make sure that they have already explored all the possible viruses found in a single species.
As we can see, the curve is asymptotic. So, analyzing 500 samples is enough (it is not worth to spend more money to obtain 4% more species)



Anthony et al. *MBio*. 4(5):e00598-13. 2013

If we extrapolate to all mammalian species, there should be 320000 mammalian viruses. In NCBI we only have 5000 complete viral genomes. So it’s not complete

Estimating Sample Richness

We have to estimate the complexity of the sample, how many species do we have and which is the composition (if one species is more abundant).

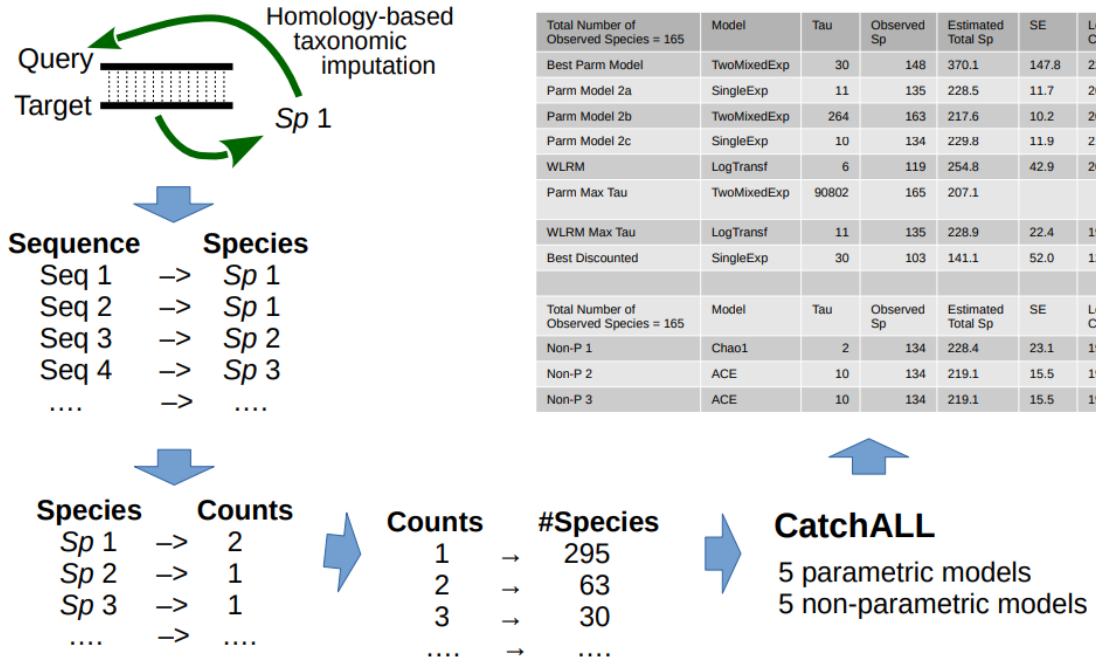
So, it's similar to accounting how many times the transcripts appear in a transcriptome assembly, to quantify the expression of the genes. But here, we are looking at the composition of the sample. Not only taking into account the coverage but also the differences in coverage.
It is not the same having 50% of A species, 25% of B and 25% of C (which is 100 reads distributed across the 3 species) and having 95% of A, 4% of B and 1% of C.
The composition of this ecosystem is going to be very different.

We can detect the 3 species, but the complexity of the ecosystem is different.

So, we calculate the Richness the same way we calculated the count distribution for the k-mers. We just count how many times each distinct k-mer appears and then we just plot how many times a k-mer that appears once, twice, 10, 20 times… is in the sample.

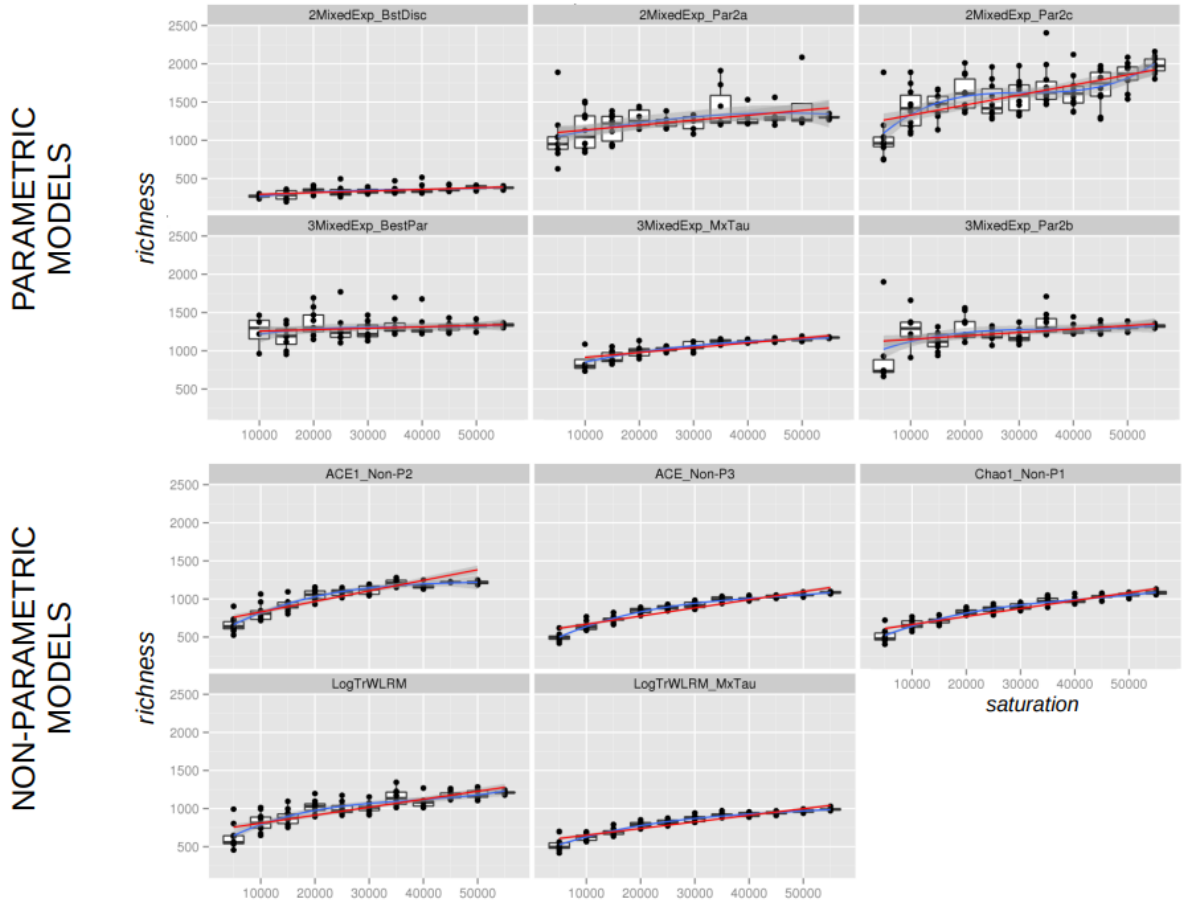
So, here we can do something similar.
We have the query, which is the sequence from the sample and we align to the DB.
The target in the DB has annotated a species and, hence, we can just transfer the species based on homology to the query.
For every single query, we will obtain a species. So, we count how many times each species appears in the metagenomic sample.
Now we count how many times we have a species appearing 1, 2, 3, 4 times.

There are programs like CatchAll that take those distributions and use different models to calculate the complexity of the sample.



Total Number of Observed Species = 165	Model	Tau	Observed Sp	Estimated Total Sp	SE	Lower CB	Upper CB
Best Parm Model	TwoMixedExp	30	148	370.1	147.8	222.7	893.4
Parm Model 2a	SingleExp	11	135	228.5	11.7	209.4	255.9
Parm Model 2b	TwoMixedExp	264	163	217.6	10.2	201.0	241.8
Parm Model 2c	SingleExp	10	134	229.8	11.9	210.3	257.7
WLRM	LogTransf	6	119	254.8	42.9	201.9	383.4
Parm Max Tau	TwoMixedExp	90802	165	207.1			
WLRM Max Tau	LogTransf	11	135	228.9	22.4	197.7	289.6
Best Discounted	SingleExp	30	103	141.1	52.0	121.0	445.1
Total Number of Observed Species = 165	Model	Tau	Observed Sp	Estimated Total Sp	SE	Lower CB	Upper CB
Non-P 1	Chao1	2	134	228.4	23.1	196.7	291.8
Non-P 2	ACE	10	134	219.1	15.5	196.1	258.8
Non-P 3	ACE	10	134	219.1	15.5	196.1	258.8

When we have a program that can use different parameters (different models) and provide different results for each parameter, how do we know which parameters we need to use in order to assess the richness of my sample?
We have to do a saturation analysis.
Each plot shows the results of different parameters.
We will choose the plot that has the best regression, the least dispersion… (Chao_1 in this case).



Unit 4. Modeling Sequences

We have seen where the sequences come from and how we can use them in order to get the base data from which we can start the next step.
We already have the assembled genome or transcriptome and now we have to focus on the annotation step.

But prior to the annotation step, we have to generate models with the data. As many programs that predict features annotated over genomic sequences depend on models.
For instance, to annotate the splicing sites we first need to build a model that predicts the splicing sites.

Sequences as strings

So, we have to model sequences and we will go from the simplest to the most complex models. The simplest model we can build for a sequence is the consensus sequence.

How do we make a consensus sequence? Just select the nucleotides that have the highest frequency.

For example, here we have a series of sequences that correspond to a splice site.

1	2	3	**4**	**5**	6	7	8	9
C	A	G	G	T	A	C	C	C
G	A	G	G	T	G	A	G	A
C	T	G	G	T	G	A	G	G
T	A	G	G	T	G	A	G	T
C	A	G	G	T	C	T	G	T
C	T	G	G	T	G	A	G	C
C	A	G	G	T	A	A	G	T
C	**A**	**G**	**G**	**T**	**G**	**A**	**G**	**T**

If we want to assess the accuracy of this model using the sequences we used to create this model, we can see that there is no sequence in the training set that matches the model.
So, we have a lot of false negatives (all the sequences) if I am looking at this exact pattern.

Let's make the model a little bit more flexible and we are going to allow one mismatch. I will still have 5 false negatives.
If I allow 2 mismatches, I will have less false negatives but more false positives.

The problem of this model, apart from being too simple, it is not modeling the composition. We do not know if there are 100% or 60% C in the first position of the consensus sequence, for instance.

We can improve the consensus sequence by adding the composition. The only way we can do that in a model is taking into account a regular expression.
So, we will define patterns that will define the nucleotides that are present in each position.

What is a REGEX

A regular expression is a pattern.

We have to define the models in a way that, using a formal language, we can describe in a unique way what we want to model. To do this, we need a sequence of symbols, which define the nucleotide strings.

We use a finite alphabet (limited number of symbols) that can be used to define any nucleotide string of any size.

Grammars

The grammar defines how we can combine the symbols to create the string.

Grammars hierarchy

There is a hierarchy of grammars that defines 4 general types.
Each grammar type is linked to an automata.
The more **general grammars** can produce any kind of string (unrestricted). But we do not want something that is general, we want something that is highly specific to build models. So, we can go from a real general grammar that can model any type of string to something that must model a specific type of string.
We are interested in the regular grammars in order to create those specific models for specific kinds of strings, which correspond in our case in a series of nucleotides that define a signal in our DNA molecules.

At the end, if we can transform a grammar into an automaton, we can transform this automaton into a program so that we can get a program that processes strings using that model.

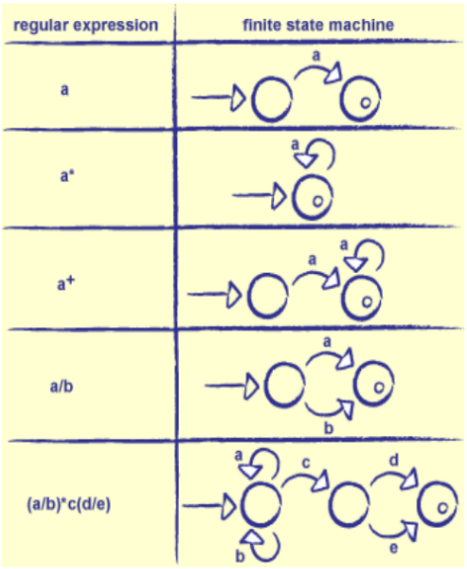
What happens with context sensitive grammar? Context sensitive means that the meaning of words depends on the context of the paragraph. So this grammar level corresponds to our language (Spanish).

If we remove ambiguity, we obtain a **context free** grammar, which corresponds to programming languages.

Finally, we have the **regular** grammars that allow us to define search patterns.

REGEX as FSA

a* = 0 or more times an “a”
a+ = 1 or more times an “a”
a/b = Have a match if it starts with a or b

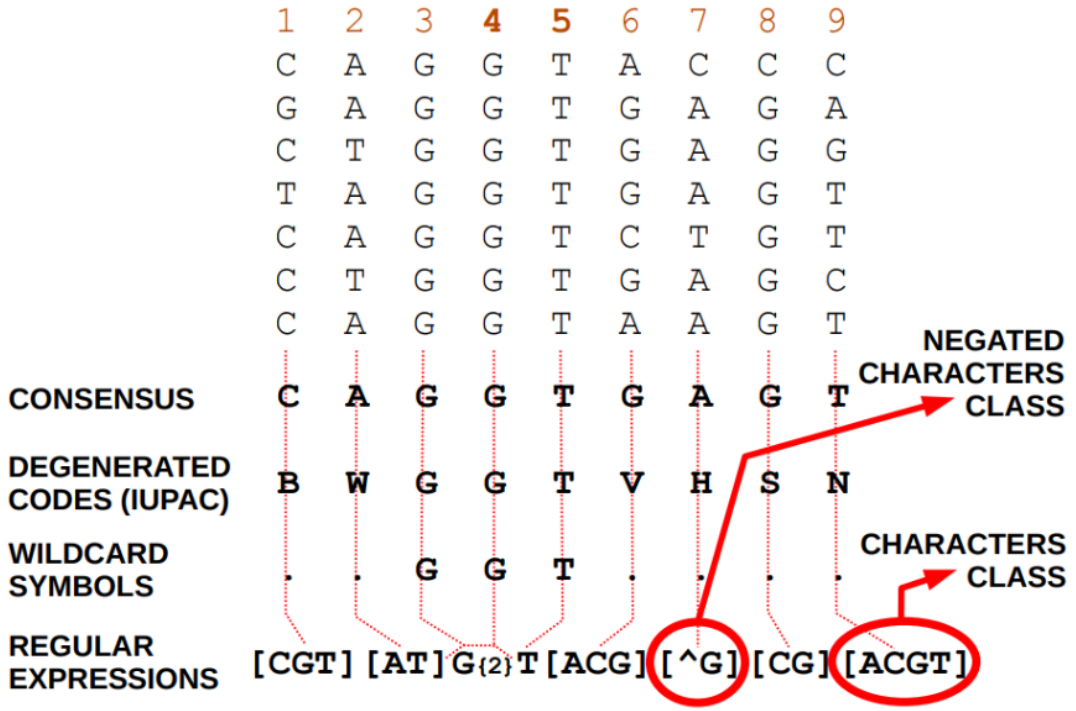


These 3 regular expressions correspond to a model for a single position.

Now, here I have a model for 3 consecutive positions:
(a/b)*c(d/e) = Start with a or b 0 or more times then there is a c and then a d or e

Regular expressions on Sequences

As we have seen, we can use the consensus sequence, we can also use the degenerated codes of the IUPAC, wildcard symbols (we only put the nucleotides with a frequency of 100%) or classes, which are an implementation of the OR.



So, the last model indicates the composition of the sequence we used to train the model. But the problem is that we do not take into account the frequency of the nucleotides.
We will take into account this in the position weight matrices.

PROSITE stores information about the patterns using regular expressions, logos…

Weight Matrices

They store the frequency or probability of a nucleotide for a certain position.
It just transforms the counts of nucleotides that appear at a given position. So, we take a set of known sequences, we align them and annotate the frequency of each nucleotide in each position.
Note that it's a special alignment because it's a position alignment.
Here we have in blue the signal and thus we anchor all the sequences along this signal and we are just aligning positions (we do not take into account if there are substitutions, gaps… but just that GT are the first nucleotides of the signal).

We will take the neighbor nucleotides that will define the signal context.
Sometimes these nucleotides are also important and they contribute to the signal.

So, here we have stacked or aligned the sequences and now we just compute the frequency of each nucleotide for each position and we create the position weight matrix.

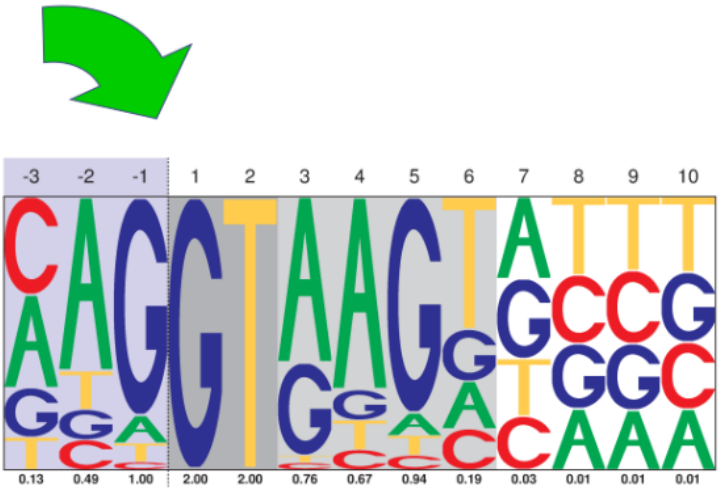
#	A	C	G	T
1	0.341	0.359	0.181	0.119
2	0.638	0.110	0.111	0.141
3	0.100	0.029	0.802	0.069
4	0.000	0.000	1.000	0.000
5	0.000	0.000	0.000	1.000
6	0.609	0.026	0.339	0.027
7	0.707	0.074	0.111	0.108
8	0.082	0.053	0.794	0.072
9	0.174	0.150	0.190	0.487

We can keep the highlight on the positions 4 and 5 (the 2 positions that define the signal).
The rows of the matrix correspond to the position and the columns to the frequency of each nucleotide.

Once we have this, how can we determine which is the optimal size of the model?
We can segment the weight matrix in different parts:

- End of the exon
- Donor Splice Site (signal)
- Context of the signal within the intron
- Several positions. Since the intron can accumulate changes at random, we will obtain the same frequencies for each nucleotide and thus the information provided by these positions is going to be low.

#	A	C	G	T
1	0.341	0.359	0.181	0.119
2	0.638	0.110	0.111	0.141
3	0.100	0.029	0.802	0.069
4	0.000	0.000	1.000	0.000
5	0.000	0.000	0.000	1.000
6	0.609	0.026	0.339	0.027
7	0.707	0.074	0.111	0.108
8	0.082	0.053	0.794	0.072
9	0.174	0.150	0.190	0.487
10	0.300	0.196	0.293	0.211
11	0.223	0.252	0.238	0.287
12	0.218	0.268	0.243	0.271
13	0.222	0.245	0.255	0.278
14	0.221	0.250	0.259	0.271
15	0.214	0.243	0.256	0.287
16	0.207	0.251	0.252	0.289
17	0.215	0.246	0.250	0.289
18	0.214	0.245	0.253	0.288
19	0.215	0.235	0.260	0.290
20	0.213	0.237	0.264	0.286
21	0.216	0.237	0.260	0.288
22	0.214	0.240	0.256	0.290
23	0.217	0.239	0.257	0.287



Pictogram for Real Donor Splice Sites

We can look at the set of frequencies or create a pictogram or logo to see the information in a more clear way:

- **Pictogram:** The size of the letters are proportional to the weight or frequencies and the most frequent nucleotide is always on top.
  So, CAGGTAAGTATTT corresponds to the sequence that scores the best with this model.

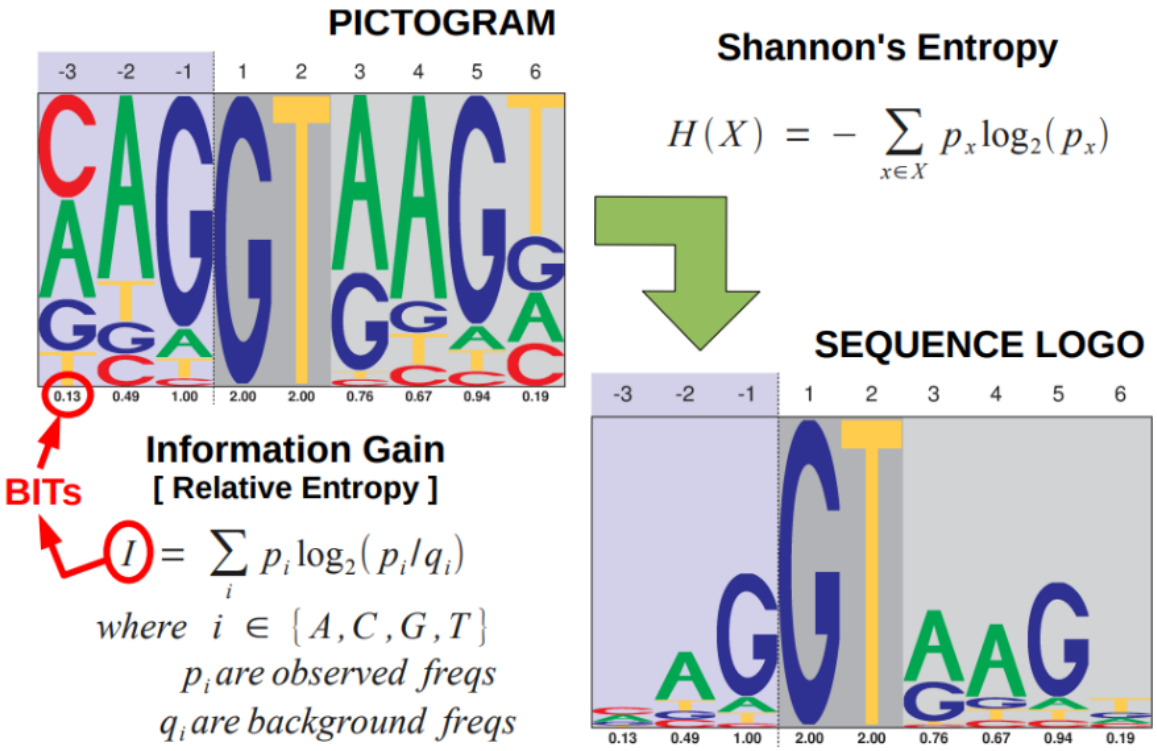
So, at what point should we cut the model? We must know the information content of the sequence.
We have already seen the Shannon's Entropy, which tells us which is the maximum amount of information of a position in a sequence, taking into account the size of the alphabet.
In our case, we have 4 nucleotides.
Each position of a nucleotide sequence can contain as much as 2 bits of information.

But this is telling us how many bits a single position can have but it is not telling the amount of information that a given position has. So, we can compute the information content (I).

Remember that the Shannon's Entropy is the general amount of information for a given string and here we have the relative entropy (I) that depends not only on the observed frequencies but also in the ratio between the observed (p_i) and the expected (q_i) frequencies that we adjust to a background model.

Any background model can be used. For instance, if I am modeling donor sites, the best background model will be the frequencies obtained from non donor sites sequences.
But this is biology is hard to define.

The simplest background model we can use is the random sequence model.



Note that we can also define the information content as Bits.
The most important positions for the model are the GT.

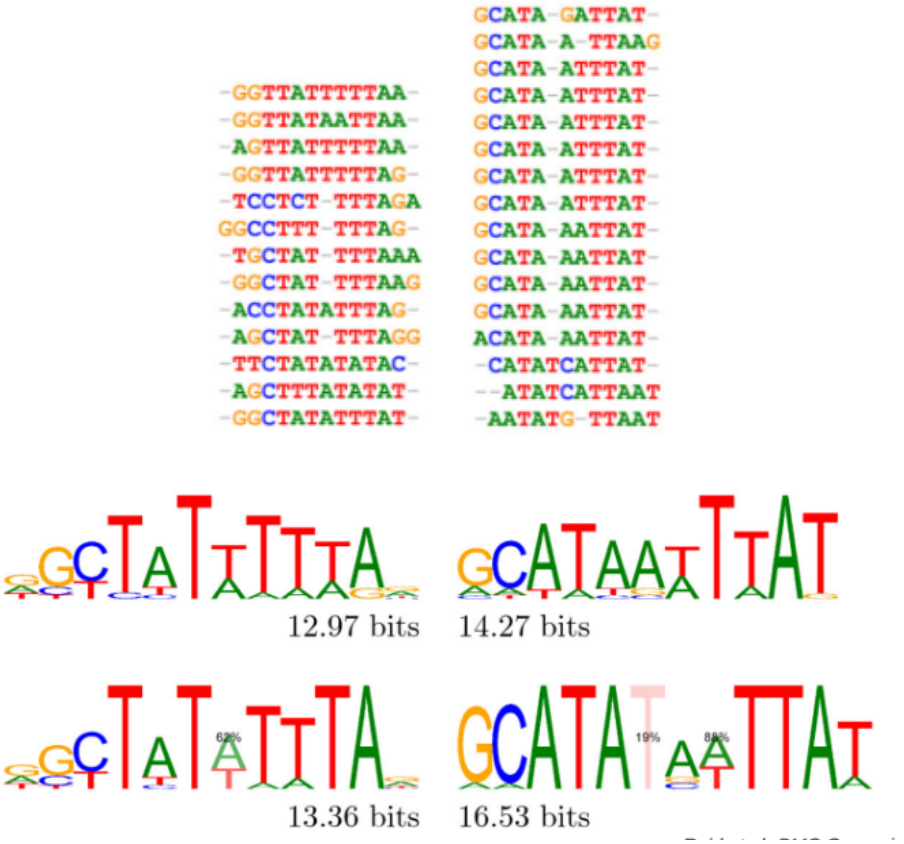
So, we can take into account the amount of information on each position to adjust the pictogram and transform it into a sequence **logo**:

- The frequencies are still there but we adjust the size of each position to its bits.

This highlights the most relevant nucleotide positions in that model.
We can't only rely on the GT (this sequence appears by chance every 16 nucleotides), but also in the context. So, the signal is defined by many positions.

Gapped PWMs

In principle we do not use gaps. There were some attempts to include gaps in the models in order to improve them. But we have to include a few gaps to improve the model (more bits in the model).
So, we put in a lighter color the nucleotides present in some gaps.

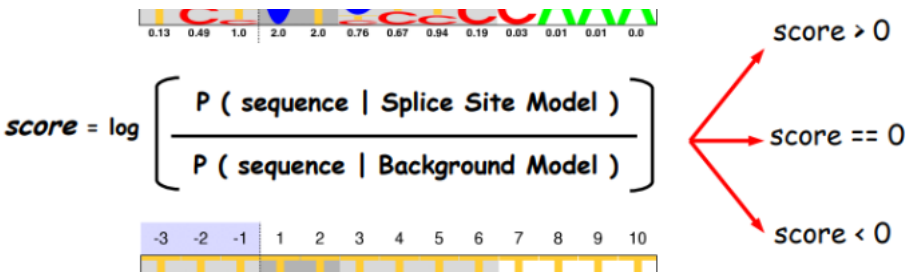


Sometimes the effort is not worth it.

Log-Likelihood Ratio

We can transform the frequencies or probabilities into log-likelihoods, which help us to decide if a position looks more like the model or the random sequence.
So, having the probability of each position of our sequence with respect to the model and with respect to the background model (random model, for example), we can just calculate the ratio and transform it into a logarithm:

- If the ratio is bigger than one, we obtain a positive value. The sequence fits the splice site model, for example.
- If the ratio is smaller than one, we get a negative value



This requires some adjusting. Depending on the signal we can have low scores.
So, we can have a lot of false negatives.
Thus, we will need to assess the accuracy of the models (we will see this later).

Probabilistic Models

In the PWMs we take into account the composition, the frequencies but not the dependencies between nucleotides.
So, each position in the matrix is independent from the other positions in the matrix.
We do not know if every time we have a “T” in the second position it depends on the existence of a “G” in the first position.

So, in the PWM model we assume that each position is independent from the others.
In order to take into account those dependencies, we have to use the probabilistic models.

Probabilistic models assess a different question. When we have assessed the likelihood, we are checking if the sequence was matching the model.

Which is the model that fits better to the sequence? The sequence matching the model? We just compute the log-likelihood ratio and if we obtain a positive value, then the sequence matches the model.

The problem is finding which is the model that matches sequence? The model matching the sequence? Finding the probability that the exon model matches the sequence we are looking at. We use the log-odds Ratio.
This log-odds Ratio is difficult to calculate. So, we can decompose into the sum of the likelihoods (easy to calculate) and the prior probabilities.

The problem is that we do not know the prior probabilities. We have not explored the space sequence enough to know which sequences can be exons or not.

Given a sequence, which model fits better to ?

$$\text{log-likelihood Ratio} \rightarrow R = \log \frac{L(S|M_{exon})}{L(S|M_{intron})}$$

Which is the probability that a given model is true given the observed sequence ?

$$\text{log-odds Ratio} \rightarrow R' = \log \frac{P(M_{exon}|S)}{P(M_{intron}|S)}$$

$$R' = \log \frac{P(M_{exon}|S)}{P(M_{intron}|S)} = \log \frac{L(S|M_{exon})}{L(S|M_{intron})} + \log \frac{P(M_{exon})}{P(M_{intron})}$$

Posterior Probabilities — Likelihood — Prior Probabilities

So, now the probabilities in each position are not independent.

$$P(S) = P(s_n|s_1 s_2 \ldots s_{n-1}) P(s_{n-1}|s_1 s_2 \ldots s_{n-2}) \ldots P(s_2|s_1) P(s_1)$$

So, given a first nucleotide, we can look at the transition matrix and look at the probability of finding the next nucleotide given that the first nucleotide.

Markov

We can use the Markov chains models to take into account the nucleotide before, the 2 nucleotides before… With this we define the order of the MC:

- Order 0 has no dependencies. So it has the same probabilities as a PWM.
- Order n: How many positions we take into account for the dependencies.

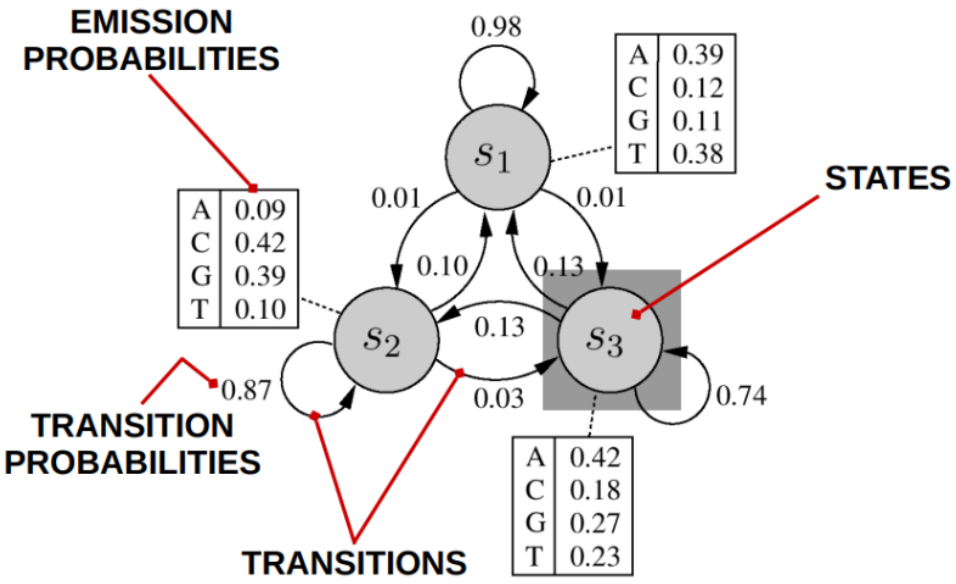
So, we just take into account some of the nucleotides before, not all of them.
If we have an order of 5, how many different probabilities do we have to calculate?
Since we have 6 positions, then 4^6

If we have an order of 100, we will not need to calculate all those probabilities, because some of the combinations appear more than once.

HHM

We have the non-observable states.
The nucleotide sequence is made of nucleotides (they are the observable states) but we do not know what those nucleotides correspond to (non-observable states). For example, signal or no signal, having more or less GC composition…
So, the non-observable states are the things that we don't know, that we can not guess.

At the end, we create a graph that contains the states, transition probabilities and emission probabilities.
Here we have a **sequence content sensor** based on HMM. The sequence content sensors allow us to estimate sequence composition.
Here we have the probabilities of the sequences that are used to train the 3 states (S1, S2, S3). So, we have a model with 3 states that define 3 levels of GC content (poor, medium and rich).

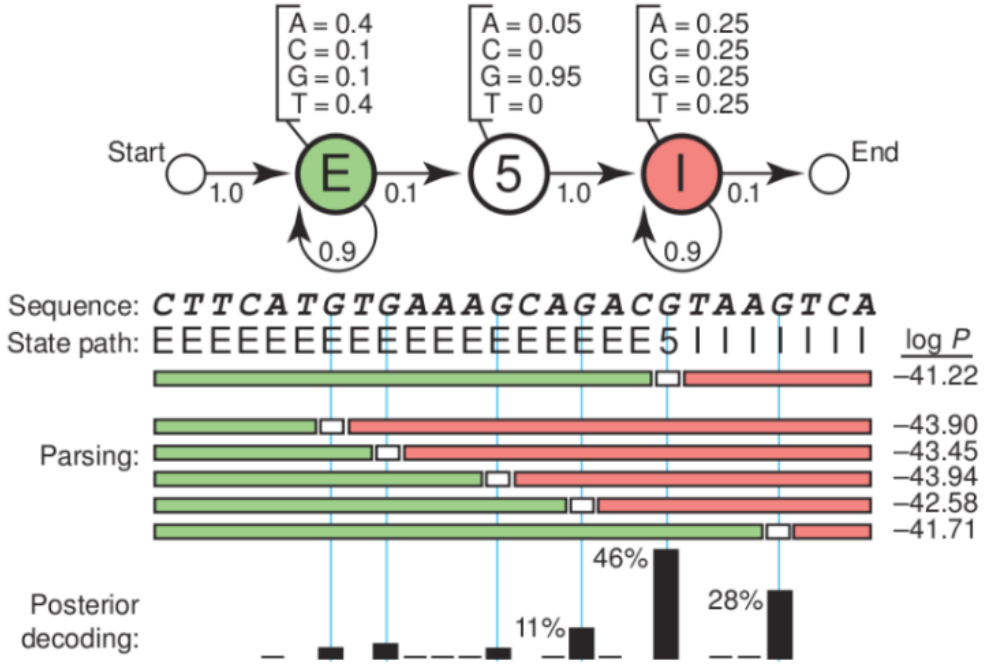


The transitions tell us the probability of switching from one state to the other. So, I am in a rich GC region and at some point I swap to a GC poor region.

Note that there are also self transition probabilities, which are way higher.
They allow us to define if the rich GC regions are longer than the poor GC regions, for example. We define the extension of the regions!

We can use the same approach to model **signal sensors.**
For example, I want to find where the splice site is defining a donor site where we have the switch from a coding region and an intron. So, finding the donor site is finding the transition between the exon and the intron.

Here we have a simple model that now it's not focusing on the composition, length… of the splice site but finding where there is the splice site.

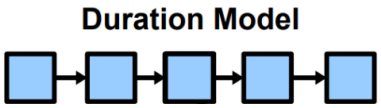


Once we enter into the signal state, then the probability of entering the intron is 100%.
We can also add starting and ending nodes which allow us to combine the models.

As we said, we know the observable states (nucleotides) but we do not know which GT is the donor splice site. Using this model, we can traverse the sequence using different paths that give us different scores.
So, the traversal returns a score.
Thus, each path will have a score and we will select the optimal one, which has the lowest "logP".

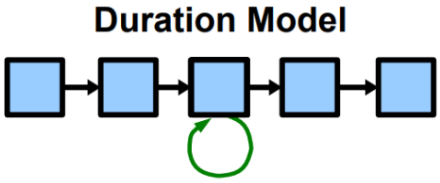
Using a graph we can model any HMM and depending on how we model the transitions, we can define different types of signals.

The simplest model is the **Duration Model**.
Here I have a state that defines a single position. How can we model a signal that is defined by more than one position? I can just model each position separately.
Here I have a 5 position model.



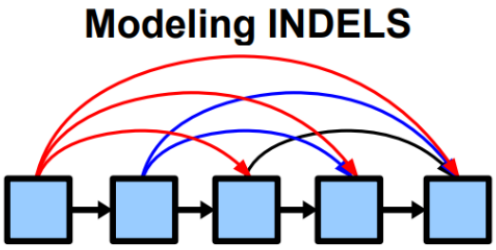
But what happens when a signal has a variable length.
Imagine that the signal can have a length of 5 or 6 nucleotides. This duration model that is based on 5 fixed states will only fit the signals that have 5 positions.
So, the signals that have 6 positions are not going to be well predicted with the fixed number of states model.

For this reason, we include the self-transitions, which allow us to expand the signal.



What happens with indels? When I am defining positions, I have to take into account that in some cases a position is skipped. So, from a given node in the graph we can take 2 paths:

- Going to the next node
- Skipping the next node



If we have to take into account all possible indels, this means that we have to add a lot of edges:

Total_Edges = (num_states -1)! + 1

The more edges, the more transition probabilities we will have to calculate.

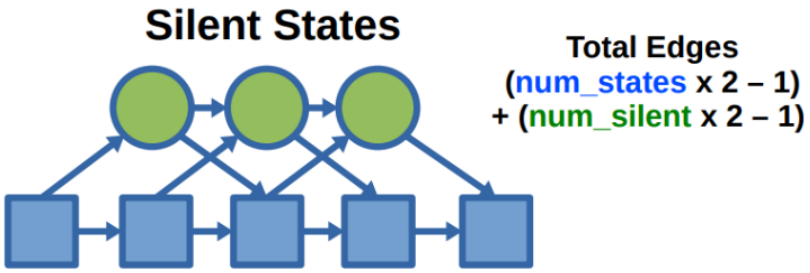
So, is there any model approach that avoids calculating so many transition probabilities?
Yes, we can use the silent states.

Instead of adding edges into the graph, we add nodes that are not emitting positions.
All the squares are position nodes that have an emission probability (with the probabilities of having an a, c, g, t) and the balls are the silent states that don't have emission probabilities.

Every silent state has 2 possible exit edges. So, we have 2 transitions:

- To the next position
- To the silent node

So, we can model a whole gap or a gap of one nucleotide depending on how many silent nodes we go through.



This allows us to generate profile HMMs.
PFAM uses HMM.

Unit 5. Gene Finding

We are going to combine the signals to find higher structures like genes.

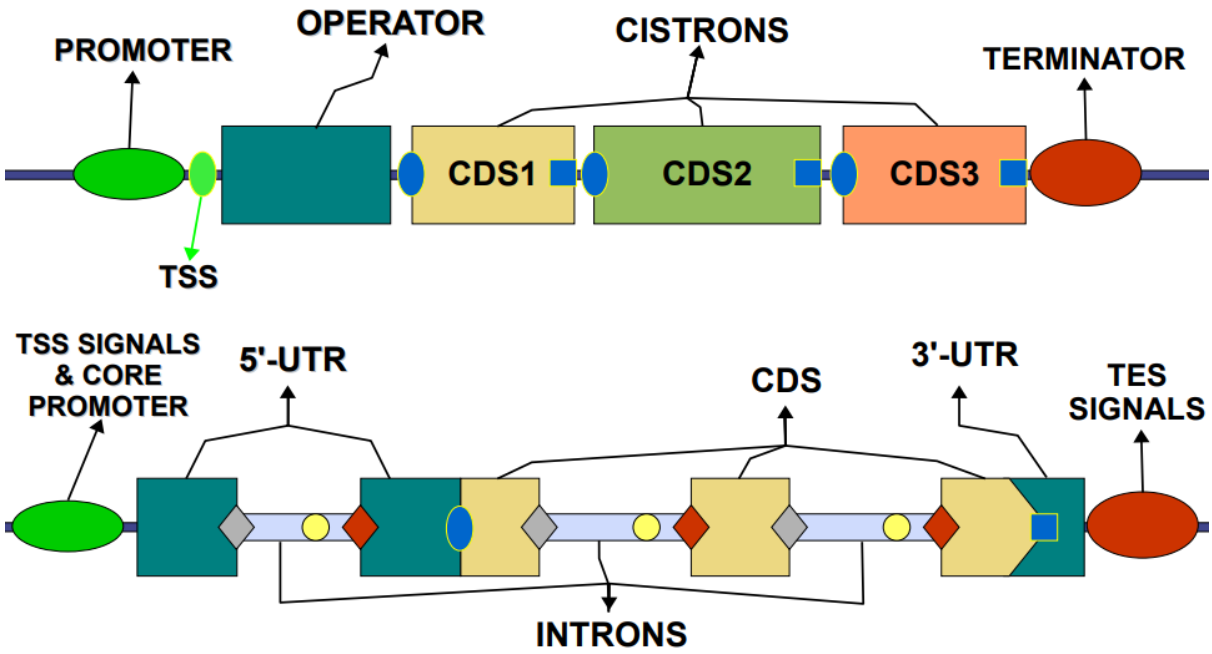
So, the first thing we must do is to define a gene.
Genes have distal 5' TSSs and regulatory regions and share their genomic sequences with long and short noncoding RNAs encoded on both strands.
So it's not only the coding region.

The gene finding problem

It must be species specific.
In prokaryotes there are no introns (continuous coding regions) but they can be translated in packs (they are clustered together into a unique transcription unit, the cistron).

In eukaryotes we have introns and the polymerase has a tail that collects splicing factors and many things that start processing the transcript, so that they can process the transcript removing the introns... also, there are a lot of transport and regulatory elements in the middle.

Also, there are different genetic codes for different organisms. Also inside the same individual (mitochondrial and nucleus).

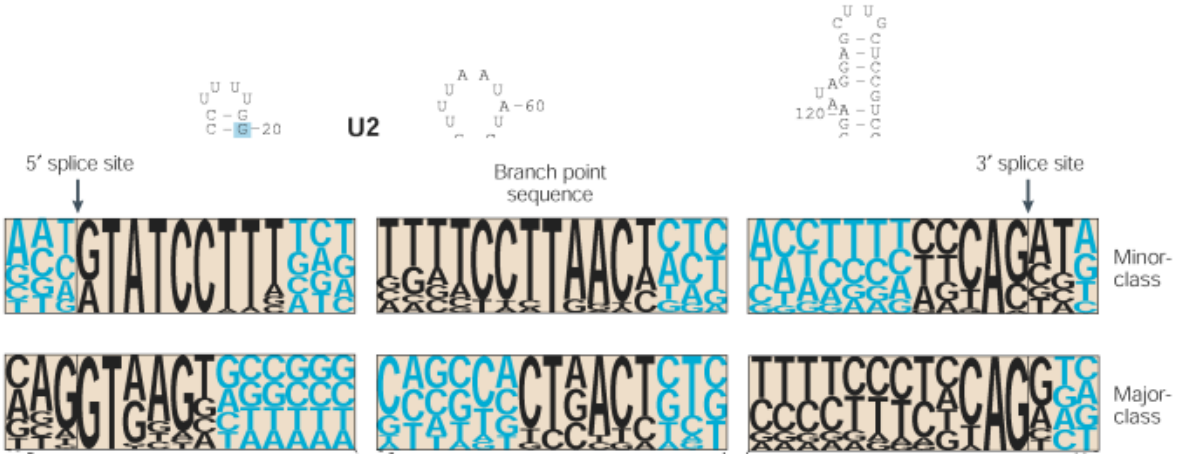


In eukarya we have a lot of transcripts.

Also, the molecular machinery of the cell is different in different species.
There are 2 splicing machineries (U2 and U12) that recognize different splice sites.

We normally use the U2 that recognizes the 95% of splice sites and thus there are some splice sites that will be wrongly predicted.
Why are there 2 machineries? Because it is a form of regulation even though it is energetically demanding.

For the non-canonical donor splice site (U12), there is more conservation at the beginning of the intron. So, we have to look at the context to detect if it's a minor or major class.



Viral Genomes Classification

There are a lot of genomes:

- ssRNA, dsRNA, ssDNA, dsDNA that can be + or -

Most viral genomes don't have all the machinery to replicate because they use the machinery of the host. Moreover, there are a lot of overlapping reading frames.

Computational Gene Finding

The easiest thing is to translate DNA into a protein using the genetic code.
We can just use a translation tool.

It will return the protein but it does a lot of assumptions:

- Correct starting codon
- No introns
- Correct stop codon

Annotating Coding Segments

We can annotate the Open Reading Frames (ORF) of specific sets of sequences.
If I have a transcriptome, I can try to look for coding biases (there are differences in the 3' and 5' UTRs and the coding regions).
Thus, we can have tools (ORF finders) that search for those biases and find the starting and ending point and then construct the protein.

But still this is not a model, we are just translating.

ESTscan can skip errors in sequencing (gaps, indels, mismatches) and get the longest ORF possible. It uses a model that considers biases between coding and non-coding.
It's a HMM.

Computational Gene-Finding

The real gene finders are based on signals and sequence biases that define coding and non-coding regions and how we can combine those coding elements into a larger structure:

- We will use signals models to find the boundaries of exons
- Sequence content biases to detect coding and non-coding.

There is a simple classification of the gene finding tools:

- **Searching by signal**: Modeling signals defining gene features (mentioned before)
- **Search by Content**: Describing sequence properties (mentioned before)
- **Gene-Finding Approaches:**
    - **Ab initio gene finding:** Tools that only use signal models and content biases to make the predictions (they don't use external information apart from the models).
    - **Homology search on annotation DB:** We have information of known genes and, hence, we can map the sequences over the genome. So, we take advantage of conservation. So, we are adding more information on the Ab initio gene finder.
    - **Comparative genomics:** We compare 2 genomes to get evolutionary conserved elements.
    - **Next Generation Sequencing evidences:** Includes the information about transcriptomes.

There is always ab initio!

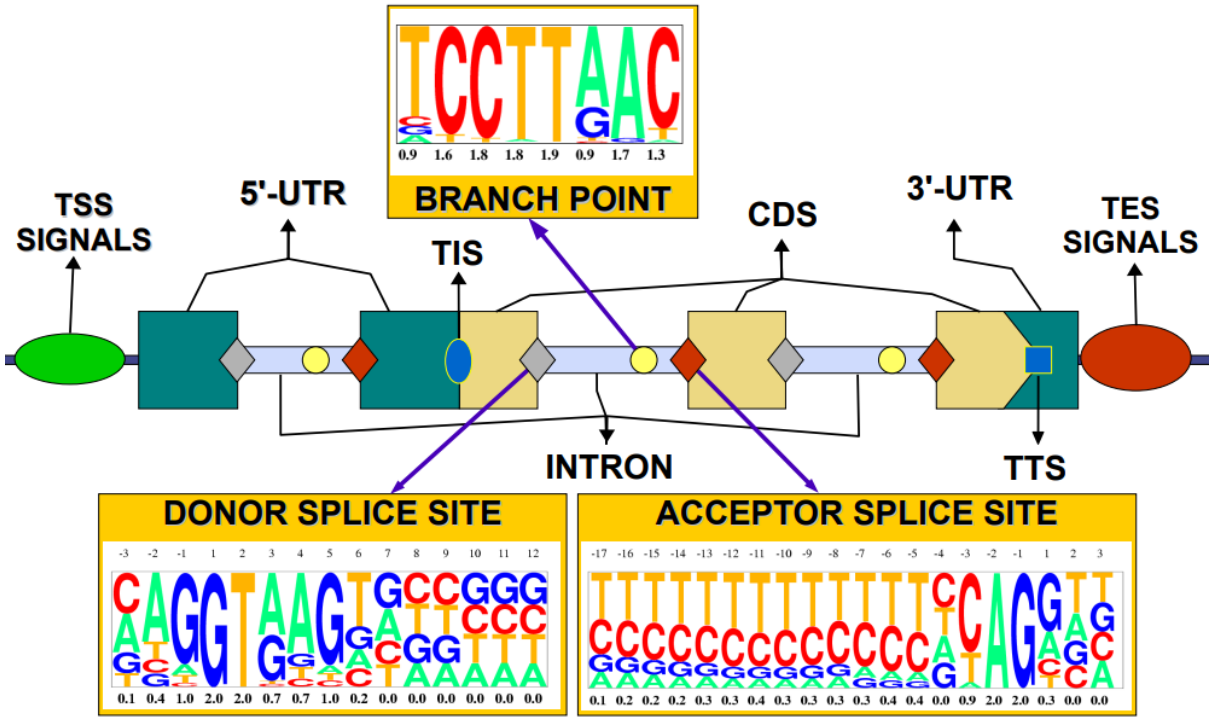
Search by Signal

Here we have 3 different models for each of the 3 signals:

- Each different signal requires its own model
    - The lengths and the composition will be different for example

Also, this model is species specific. Because there is a lot of variation between species (specially in introns). So, the models are signal and species specific.

If we apply the model in distant species, we will have a lot of errors.



Sequence Conservation at Orthologous Splice Sites

There are some differences in composition between coding and non-coding regions.
Here we have an alignment between sequences upstream and downstream of an exon. So, we have:

- End of an intron
- Acceptor site
- Exon

And:

- Exon
- Donor site
- Intron

We are looking at the conservation between orthologous splice sites. So, the conservation of the same gene in different species.

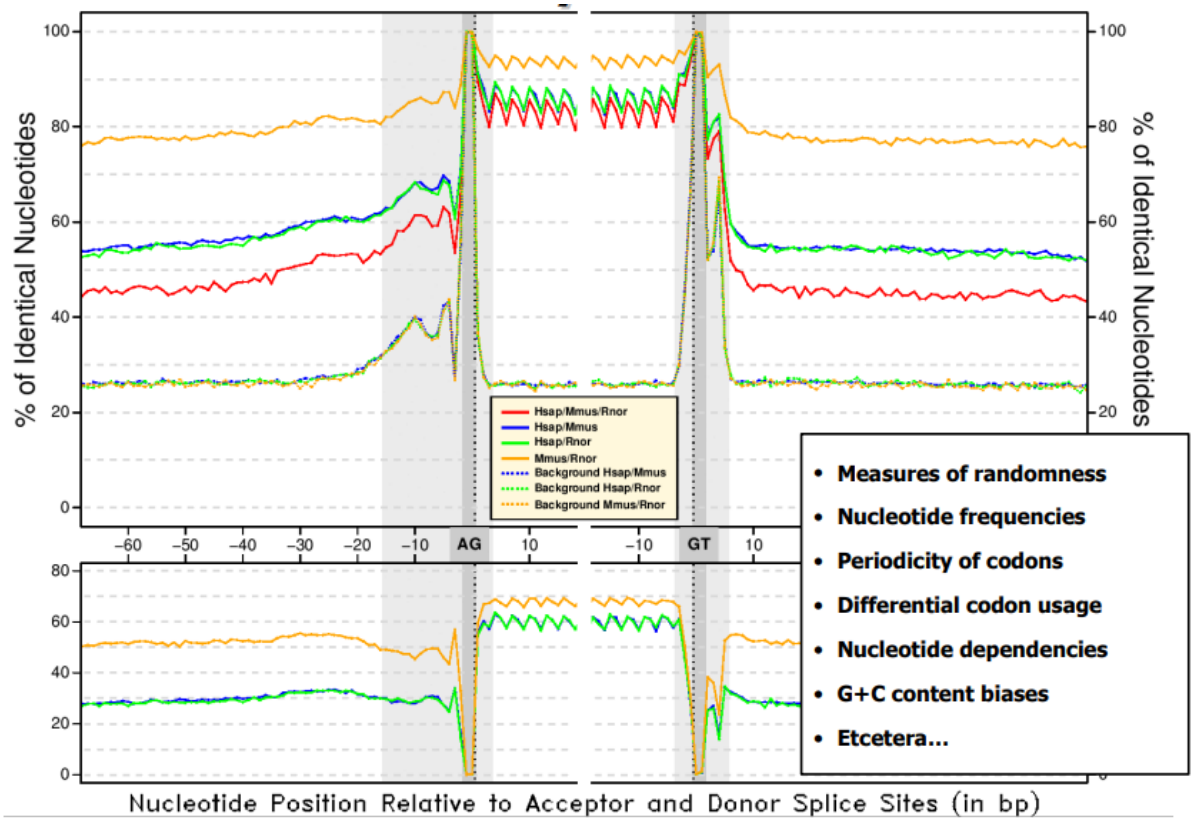
- Orange: Mouse and rat
- Blue: Human and rat
- Green: Human and mouse
- Red: Triple human, rat and mouse alignment

We can see that the conservation is higher in the coding regions.
Of course, there is more conservation in closely related species. But there is a more noticeable change in conservation.

There is a pattern similar to shark teeth. This is characteristic of a periodicity of order 3:

- We can see that there is a position that is higher, another that is in the middle and another that is the lowest every 3 nucleotides.
    - The first nucleotide is the first position of a codon, the second is the second position of a codon…

We know that the first and the second positions of a codon are more likely to be conserved than the 3rd position because of the degenerate genetic code (there are many synonymous aa that are encoded by more than one codon).

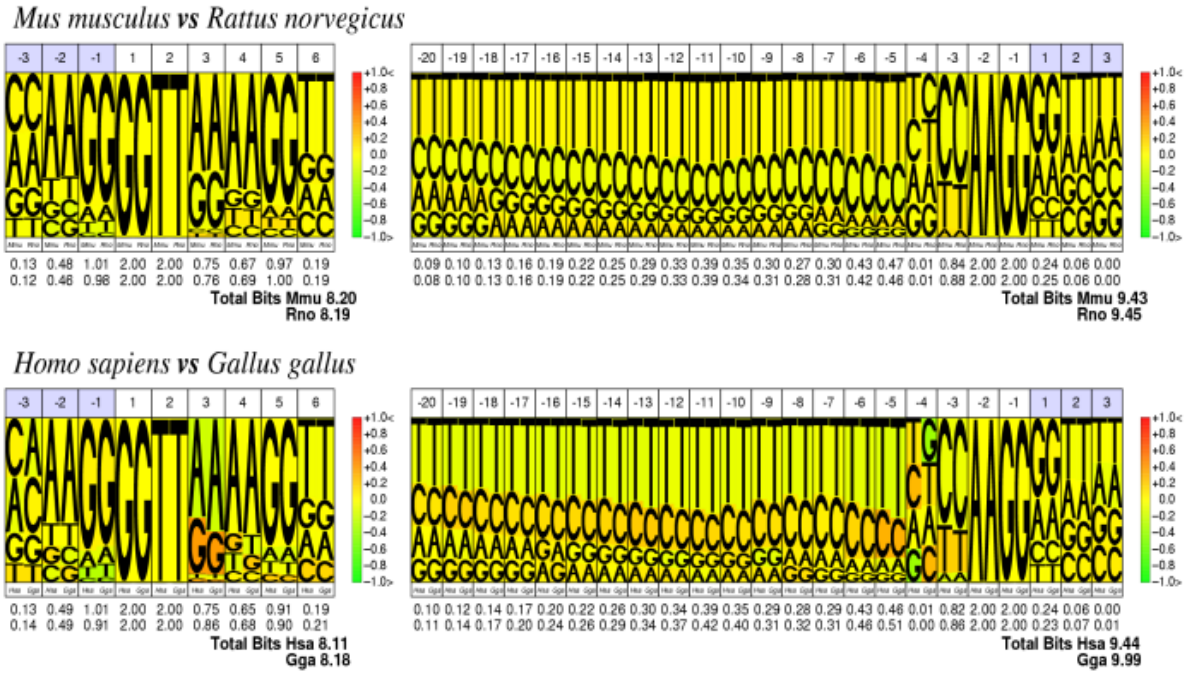


Interspecific Comparison of Splice Sites

We are talking about species specific!
Here we are comparing rat and mouse splice sites. So, for each position we have 2 nucleotides that correspond to rat and mouse respectively.

There is a color range that tells us if the position changes more in the first or second species.

When we compare close species, the splice site signals are really similar.
But when we compare distant species (human and bird), there are some positions in red or green (there are different frequencies).
Maybe we still have a gene, but the frequencies have changed. For this reason, the models should be species specific.



Intraespecific Variability of snRNAs

We also have variability on the snRNAs and this can affect the recognition of the canonical splice sites.

9th Theoretical Class

Other Signals: TSSs

Here we are still talking about the signals we can use for gene finding.
We have been talking about the donor site and acceptor site signals, which are really strong.

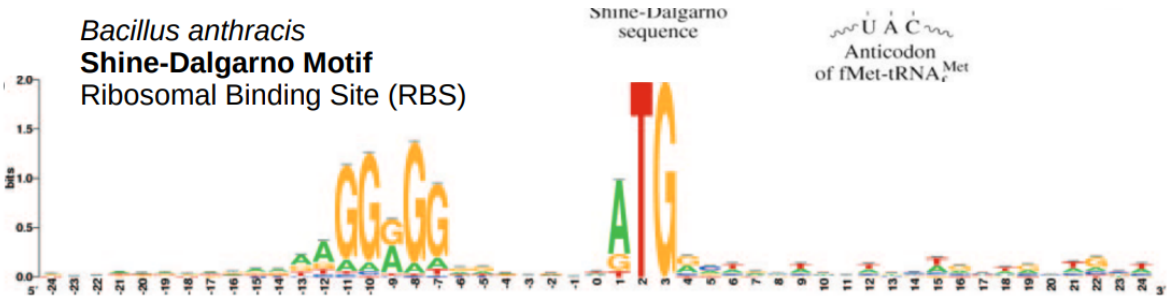
But we still need other signals to define the gene structure:

- **TSS**: Translation start site defines the first codon of the mRNA. Once we have the mature mRNA, the ribosome needs to know which is the reading frame of that RNA. Otherwise we would have a lot of defective proteins that could harm the cell.
  If we use a different reading frame, the protein is going to be completely different and also it can be truncated because it finds a premature stop codon.

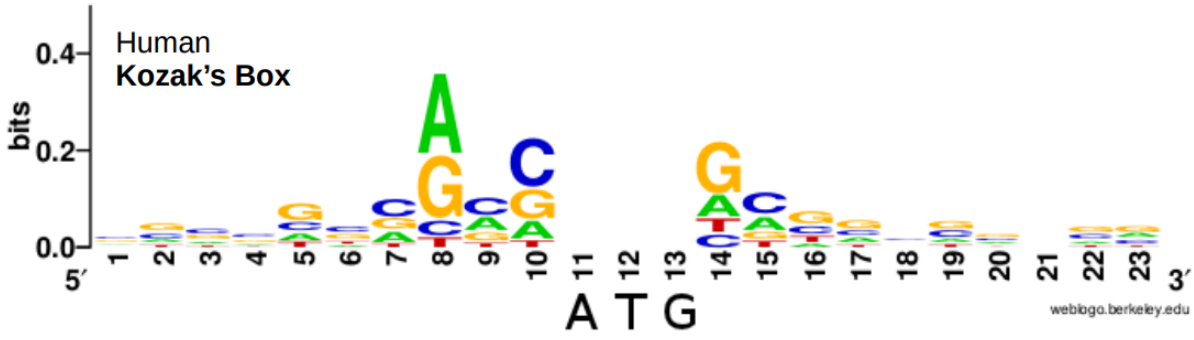
  In the image we can see that not all species start their transcripts by ATG. Otherwise, we would have 100% of times A in the first position (which is false according to the image).

  So, as mentioned, detecting the ATG by the ribosome depends on the context. It depends on another signal that is found before the starting site that corresponds to this Shine-Dalgarno Motif (in prokaryotes) or RBS.
  This motif is recognized by the 16S RNA (a ribosomal RNA).
  So it helps to set the correct reading frame of the mRNA.

  This 16S RNA is a highly conserved sequence that is used often in metagenomics to capture the mRNA of a sample.



In eukaryotes we have the 18S RNA and 20S RNA.
In this case, we have a context that is defined by the Kozak's Box.
It is not equivalent to the Shine-Dalgarno motif because in this case the context is before and after the starting codon and it is less conserved.



Other signals: TTSs

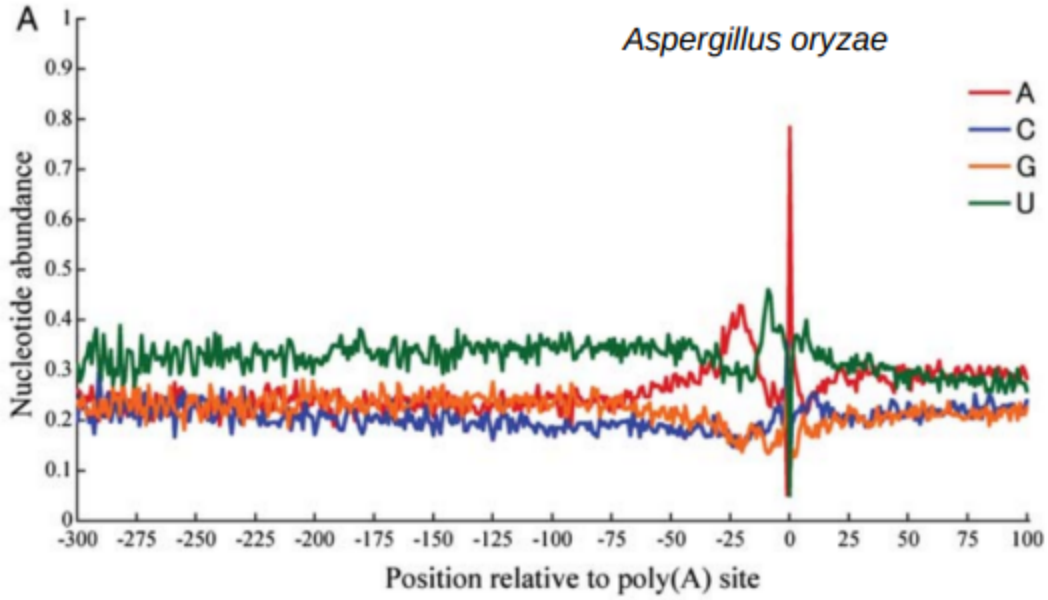
The RNA is synthesized as a copy of the DNA by the RNA polymerase and it has to stop at some point. There are some signals at the end of the 3' UTR (end of the untranslated region of the last exon) that define the transcription termination site.

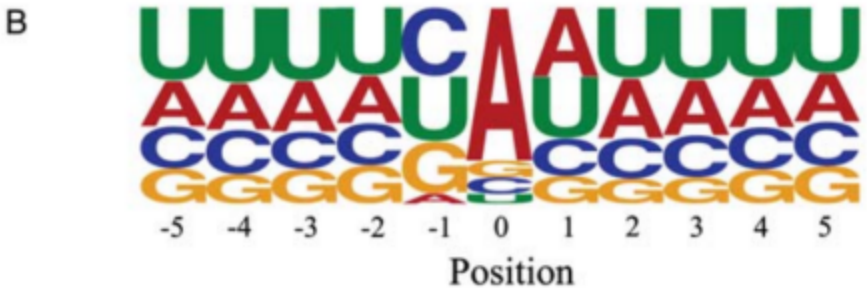
This signal is a poly-A tail (poly-Adenilation site).

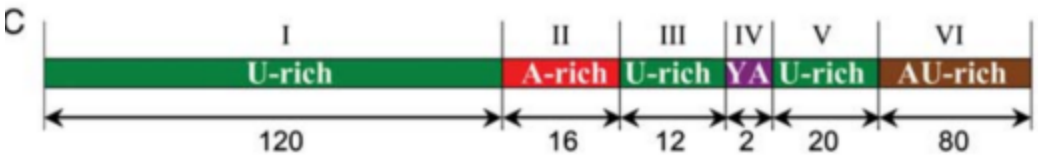
Thus, by looking at the composition of the genome we can know where the transcription termination site is found.
In this plot, we are looking at a set of sequences and we can see that the termination site has a poly-A tail (there is a bias of A nucleotides) that can be identified by a model.

In fact, we can see that there is also a higher proportion of U (context).

Note that this is a short signal and thus it will not be as strong as acceptor or donor sites.







We can combine all the models that detect all those signals and build a more accurate model to predict genes.

What problem do we face here?
Remember that we have a specificity and a sensitivity associated with any model. Thus, we can have false positives/negatives.

Here we have an example **(I HAVE CROPPED THE IMAGE, SO WE DO NOT SEE THE MOUSE ORTHOLOG)**:
We have a pair of orthologous genes (Human and mouse). More or less they have the same gene structure, since they have the same number of exons and the introns are more or less placed in the same positions.

Green boxes correspond to the coding exons and the gray boxes correspond to the 5' and 3' UTR.
We can see the donor and acceptor sites, which are also orthologous.

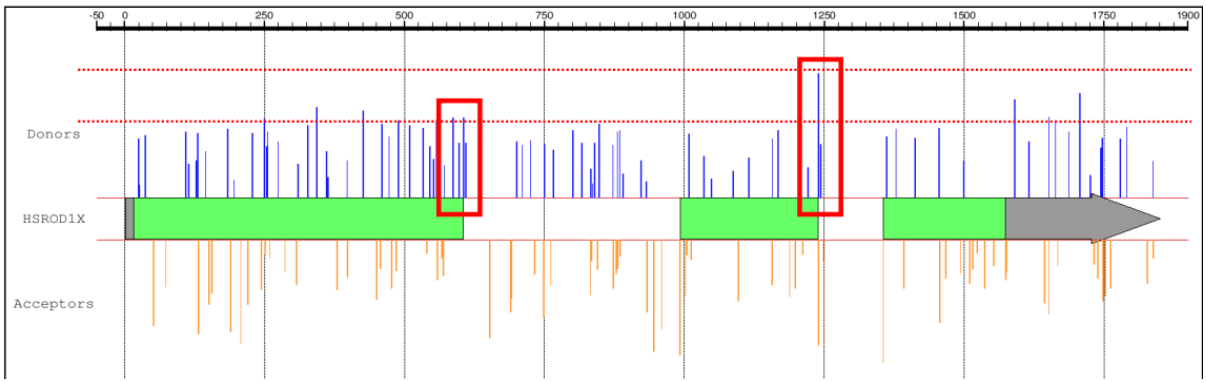
So, we have used a set of real gene annotations to define a model that finds the donor and acceptor sites in humans. We also run this for mice, since they are really similar (remember that it's better to have a model that is species specific).

The blue pics are the predictions based on a donor site model. Their height represents their score.
We can define a threshold to detect which are the donor sites:

- If we set the threshold in the first red line, we will only detect one donor site and we will miss the other.
- If we set the threshold in the second red line, we will detect both donor sites but we will also have other false positives.

Maybe those are real donor sites for another isoform. But we do not know and, hence, the model is far from perfect.

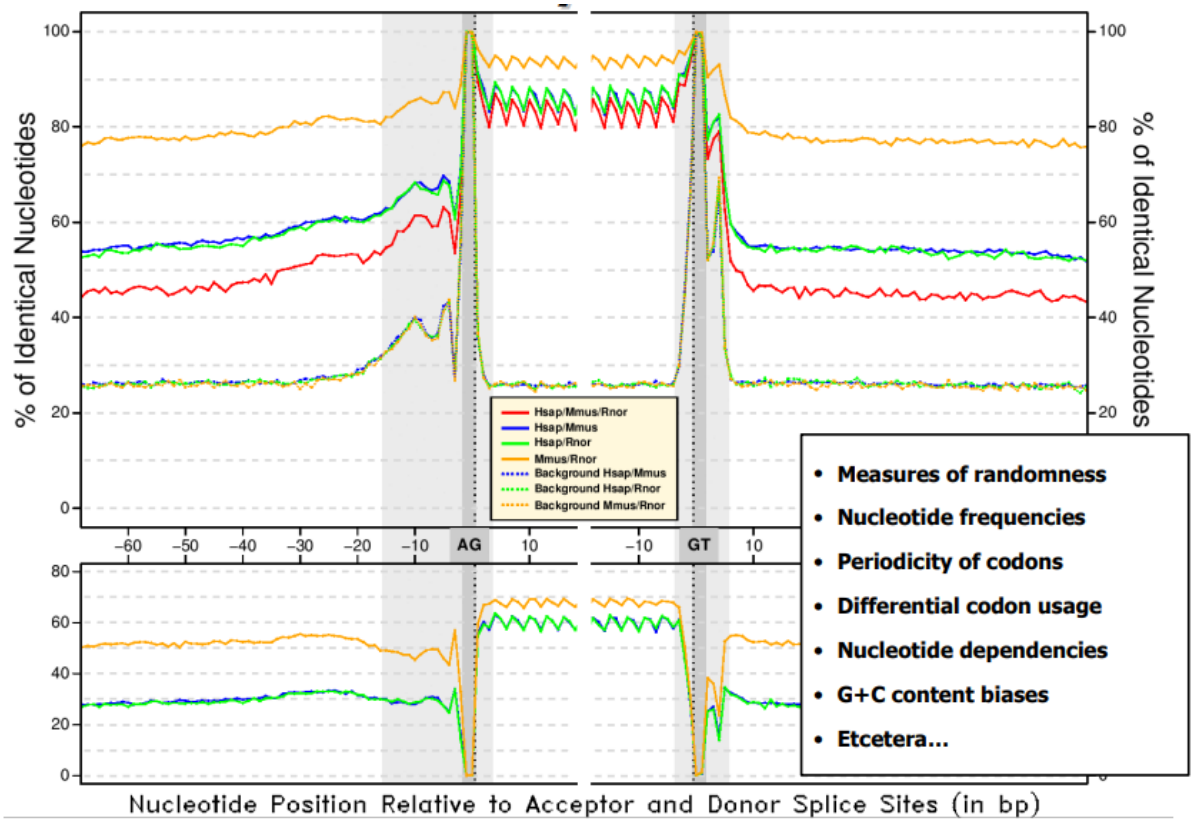
This also happens for the acceptor sites (orange pics).



So, by adding all this models we can predict better the exons:

- We have to take into account that the exons have biases (we can see the teeth pattern that corresponds to a 3 nucleotide periodicity because of the genetic code).
- Measures of randomness
- Frequencies of nucleotides
- Periodicity of codons
- Differential codon usage
- GC content biases
- Etc

Most of them are correlated with the usage of the genetic code and which are the preferent codons, in a given species, used to translate the nucleotides into proteins.



HMMs: Sequence Content Sensors

We can model sequences that look more like a coding sequence because they have a higher GC content, a periodicity different from a non-coding sequence.

The transition probabilities allow us to model the duration of the segment. This can tell the length distribution of the introns/exons.
Yeast have less number of exons and also the exons are longer than introns on average.
Humans have very long genes that have very short exons.
Depending on how we model those features in the transition probabilities, we can take into account the length of the coding regions with respect to the others.

Ab Initio Gene Finding: Geneid (EN ELS 2 PRIMERS PLOTS ESTAN TALLADES LES CADENES REVERSE)

Here we have a general diagram of how a gene finding tool works.

From an anonymous DNA sequence that is unannotated, the program tries to find all the signals in the forward strand (using position weight matrices).

In the graph, the height of the barplots is proportional to the score of the signals. We have represented in colors different signals:

- Donor site
- Acceptor site
- ATG depending on the kozak's box. Because the program "geneid" was developed to work with eukaryotic genomes (so, it has an intron mode or takes into account the donor or acceptor signals to define exons)
- STOP

The color gradient corresponds to a simple HMM of order 5 that models the dependencies of 2 aa or the 6 nucleotides defining the codons of those aa. We know that proteins have a dependency of 2 aa at the structural level.
For example, given an aminoacid of an helix alpha, there is a dependency on the previous amino acid (and this aa is also dependent on the aa before).
So, there is a bias depending on the structure in the protein.

There is a pressure to conserve the structure of the protein because otherwise the function will change. So, there is a correlation to preserve the sequence of the exons.

Color red represents coding regions and blue represents non-coding regions.
So, it has this hexamer model that takes into account this coding bias based on hexamer frequencies.

The program is strand dependent. We compute all this for the F and R strand.

Once we have predicted those signals and we have an idea of which are the regions that can be putative coding or not, the idea is that the program combines all the possible signals (summing up the scores) and predicts all the possible exons.
So, there is a combinatorial explosion.

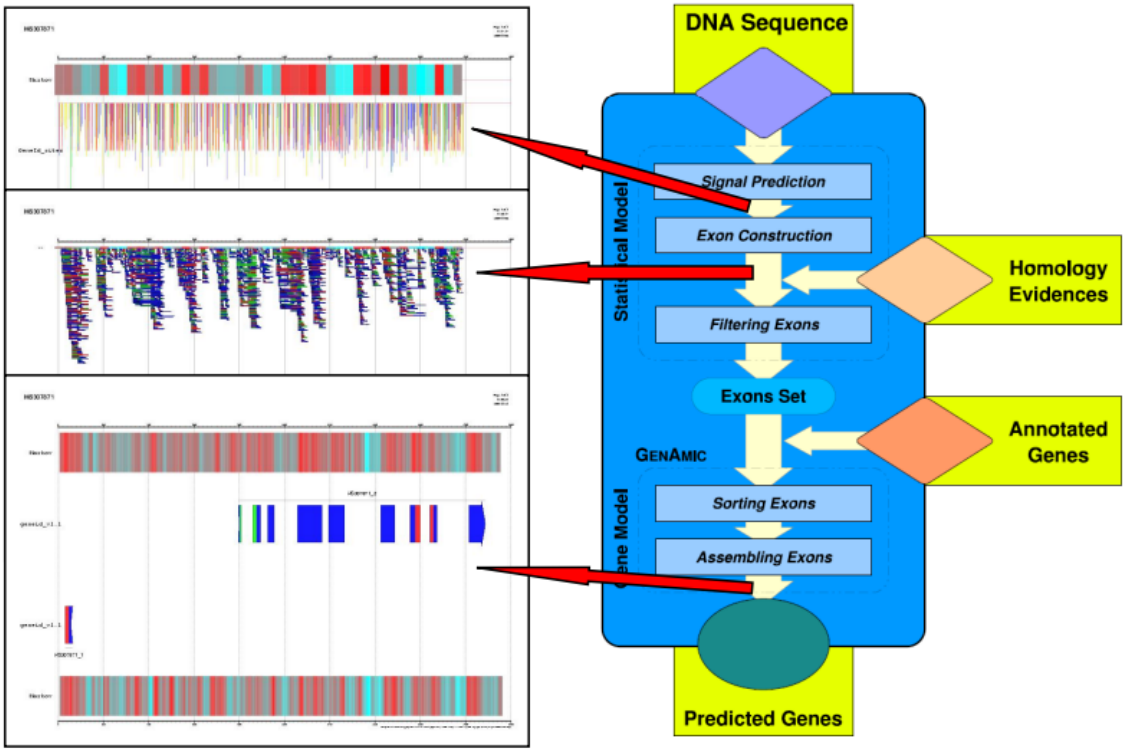
Why explosion? Because for a short sequence we will find a lot of exons.
Just imagine that in reality a gene has 30 exons (30 acceptor and 30 donor sites).
By running this program we will obtain a higher number of exons because it will consider all the possible exons (even though they are too big).
It will get the first splice site and the last one, the first splice site and the 29th…

Then it will score all the exons and use a threshold. This will filter out most of the exons.
Once we have filtered the exons that have a good score (by summing up the scores of all the signals and biases), the program will try to combine them.

It uses a gene model to combine them:

- If we have a single exon, we have a gene that contains only one exon.
- If we have multiple exons, we need to predict a starting exon that starts with ATG and ends with a donor site, an internal exon that starts with an acceptor site and ends with a donor site and terminal exons that start with an acceptor site and ends with a STOP codon.
  So, we have to predict different types of exons and how they can be combined in order to define a gene structure.

  Obviously we will have a score and we will consider the highest scoring gene structure.



We will obtain the OPTIMAL solution for the combination of best scoring exons.
These exons not only have the best score but also create a continuous coding frame.
So, when we merge all the exon sequences, we should get a continuous coding frame that can be translated into protein.
The program also takes this into account!

If we have an exon that ends with one nucleotide that is out of the last codon, the next exon will provide the 2 next nucleotides to complete the codon.

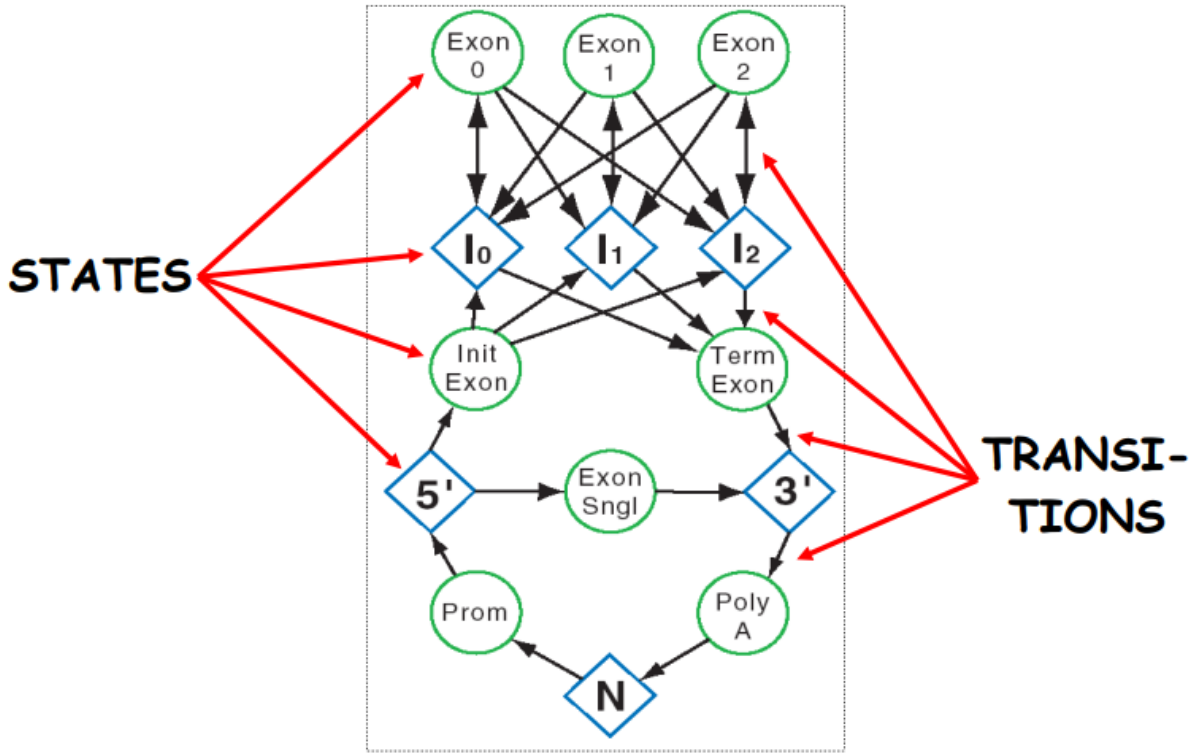
Generalized Hidden Markov Models

We have more powerful gene finding tools like “GeneScan” that apart from the HMM for the signal sensors and content sensors… they use the approach “Generalized HMM”.

What the model has defined in the states are elements of gene structure:

- Exons (start, end and intergenic)
- Introns
- UTRs
- Promotors
- Poly-A tail
- N is intergenic

This model is flexible enough to take into account any kind of gene structure.



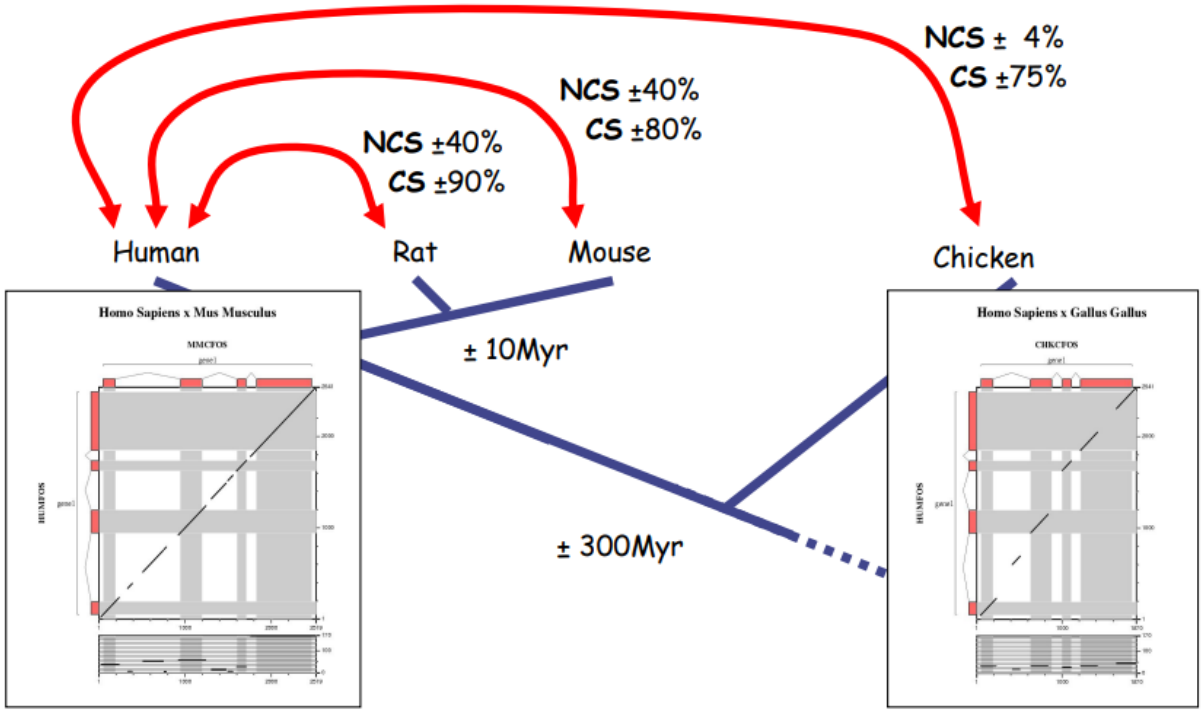
The transition probabilities can be adjusted to the average number of genes of a species.
If we have a species that has a lot of exon genes, the probability of jumping from 5’ to single exon will be 99%.

Eukaryotic Comparative Genomics

As we said, we have Ab Initio and now we are going to add further information.
Until now we were just looking at the characteristics or properties of the query sequence from which we were trying to predict the genes. Now we will try to include information about conservation.

So, the first point is to find the right species at the proper phylogenetic distance.
In the image we can see that:

- Rat and Mouse diverged 10 million years ago
- Rat and human diverged 80 millions ago (70 millions ago from human to the parent node of rat)
- Birds and mammals diverged 300 million years ago



In the plots, we can see:

- Ortholog human and mouse genes
- Ortholog human and chicken genes

We can compare the orthologs and look at their conservation:

- Red represent exons
- Gray bands represent the projections
- White boxes represent the intersection of non coding regions
- The diagonal represents the aligned segments of those 2 genomic regions

We can distinguish between coding and non-coding sequences. So, we have computed the average of conservation between coding sequences (CS) and non-coding sequences (NCS).

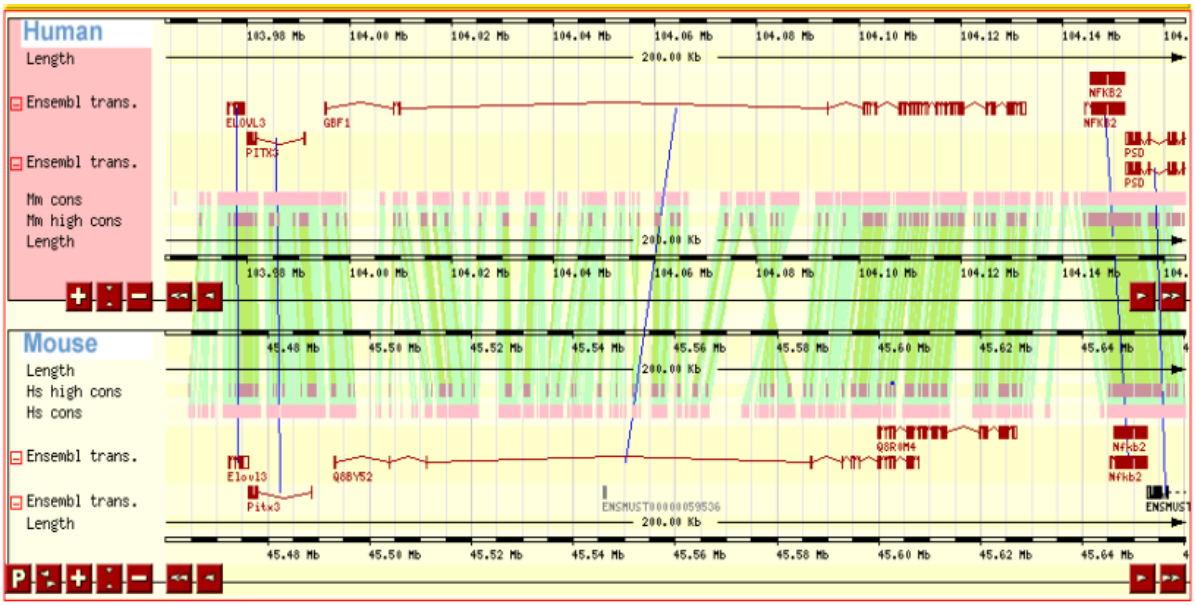
As we can see, the conservation between close relation species is higher.
Furthermore, the conservation in coding regions is much higher because a change in a coding region is probably going to produce a desfavorable advantage and therefore it will not be selected.
On the other hand, non-coding regions don't encode for a function and therefore the mutations are not going to be deleterious.

When comparing very divergent species, we can see that there is not a really big drop of conservation in the coding regions. But non-coding sequences have small conservation (there are no matches between both species).
If there is conservation in a non-coding region, this could be a hint telling us that it is an exon from an alternative splicing variant of the gene or a really important regulatory element like an enhancer.
So, probably this is a phylogenetic footprint that tells us that this is an important region that has been conserved.

We can translate those alignments into information that can be included into the gene structures predicted.

The important thing is that we must rely on syntenic regions (regions where the ortholog genes are conserved).
Here we have the human and mouse ortholog genes:

- Pink boxes: Regions that align between the genomic human region and the ortholog genomic region. A syntenic region is a region that contains a set of ortholog genes that are conserved in order.



Sequence Alignment Dynamic Programming Limits

What is the problem when facing whole genome alignments?
Depending on the aligning algorithm we use, we can have a squared or cubic cost.

If I have to compare a database (65.369·10^9 bp) against my sequence (1000 bp) using one of these algorithms, I will use 65 Tera bytes.
If we align whole genomes, we will use 3 Exabytes
So, for this reason it's very difficult to perform a perfect alignment.

For this reason, we use other approaches like BLAST (heuristic approach).
The problem of local alignments is that it can appear more than once. We will have a lot of spurious hits in different genomic regions, especially if we do not mask repetitive regions.

BLASTN vs TBLASTX

BLAST is heuristic and, hence, it will speed up the comparison of sequences because it is looking for local alignments and also its aligning from the “seeds” (keywords or common indexes that are shared between the query sequence and the targets in the database).

If we are working with nucleotides, the scoring matrix available is the identity matrix:

- + if there is a match
- - if there is a mismatch

If we are working with amino acids, we have different substitution matrices that can be adjusted to the divergence of the species. We can also use an identity matrix for the amino acids but as they share some chemical properties, there are some substitutions that are more allowed than others (so, it's better to use a substitution matrix like Blosum45).

```
Met Tyr Iso Ser Pro Asp
ATG TAT ATC TCT CCC GAC
|||  |  |   ||     ||   |  |
ATG TTT CTC AGC CCT GCC
Met Phe Leu Ser Pro Ala

                                  Amino Acid Level Score
+6   +3   +2   +4   +9   -2        Blosum45     :  +22

+++ +-+ -++ --- ++- +-+          Match/Mismatch :   +4
                                  Nucleotide Level Score
```

Note that we choose one scoring matrix for ALL the genome (this is not optimal…). We do not use different matrices for different regions.

How Blosum45 works: Blosum are derived from blocks of conserved sequences. So the 45 means that on average we use sequences that were 45% conserved.
The smaller the Blosum, the more divergence sequence we have used to build the model and, hence, the scoring allows us for more changes in the alignment (the changes are not that much penalized as in the Blosum80, for example).

If we use a different matrix, we will obtain different scores for the reason mentioned above.

Note that the score of the Proline is higher than the score of the Methionine (even though there is a mismatch in the 3rd nucleotide).
So, if we align at aa level, we can hide some substitutions in the nucleotide level.
Thus, we are allowing more changes if we are at aa level (using a Blosum in TBLASTX).

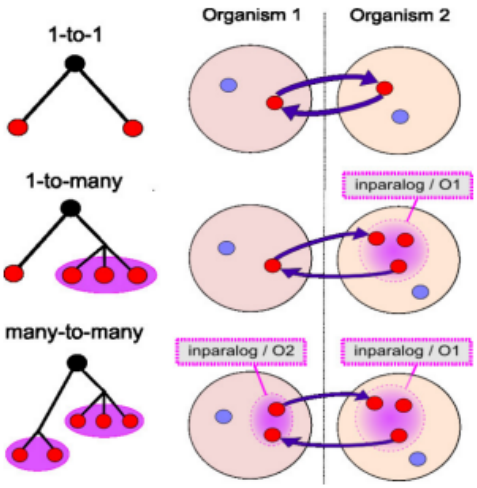
TBLASTX takes the translation of the 3 open reading frames of the forward and reverse sequence and makes the 36 comparisons of all the reading frames of one sequence against the reading frames of the other.
TBLASTX does not care which is the real frame for the sequences, but since it's a local alignment it will find the corresponding reading frames in the 2 genomic sequences.

So, overall it's better to use TBLASTX. Problem: This does not work for the non-coding regions (but since they don't have a function, it does not matter).

Fishing Orthologs

There is some filtering that we need to do because gene families can evolve differently in the species:

- In the ideal world, from one ancestor sequence gene we will have the ortholog gene in 2 species. So, there is an speciation event and therefore we have the same gene and the same function encoded in 2 different species (we know this is not true because they can evolve).
    - Maybe there is an expansion in one of the species
    - This expansion complicates the task of finding the real conserved regions through different genomes.



We are doing a BLAST that is going to provide me homology hits. But this homology hits may correspond to paralog sequences.
If I want to improve the scores of the predicted genes, I want to use orthologs, not paralogs.

One of the approaches we can use is to find the best reciprocal hits (do a BLAST in both directions):

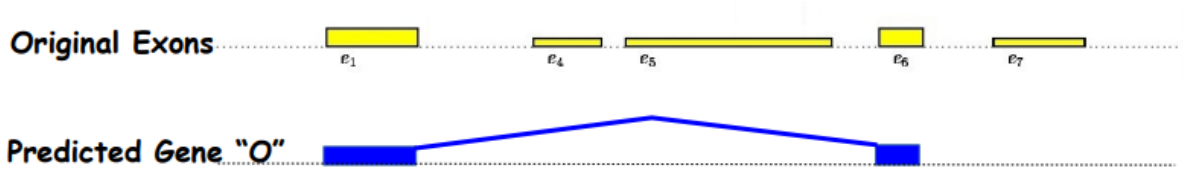
- If there is a reciprocal hit then they are orthologs

Re-Scoring Exons

How can I use that information to improve the annotation of the genes?

Let's look at this example:

- Originally we have exon 1, 4, 5, 6 and 7. These are the exons that are going to be used by the program. So, probably these exons have passed a filtering test and they are the best scoring exons for this region.
- Where are exon 2 and 3? These exons have been removed because they did not have enough score to pass the filter (but they were possible exons, initially).
- The height of the exons correspond to their score.
- Now the program has to combine them in order to find the best gene structure.
    - By taking into account the scores, the program determines that this is the structure of the gene, which combines the 2 best scoring exons.



This is Ab Initio. From the evidence of the signals, the biases… we build a set of exons that are going to be filtered and scored and we will combine them to find the optimal gene structure.

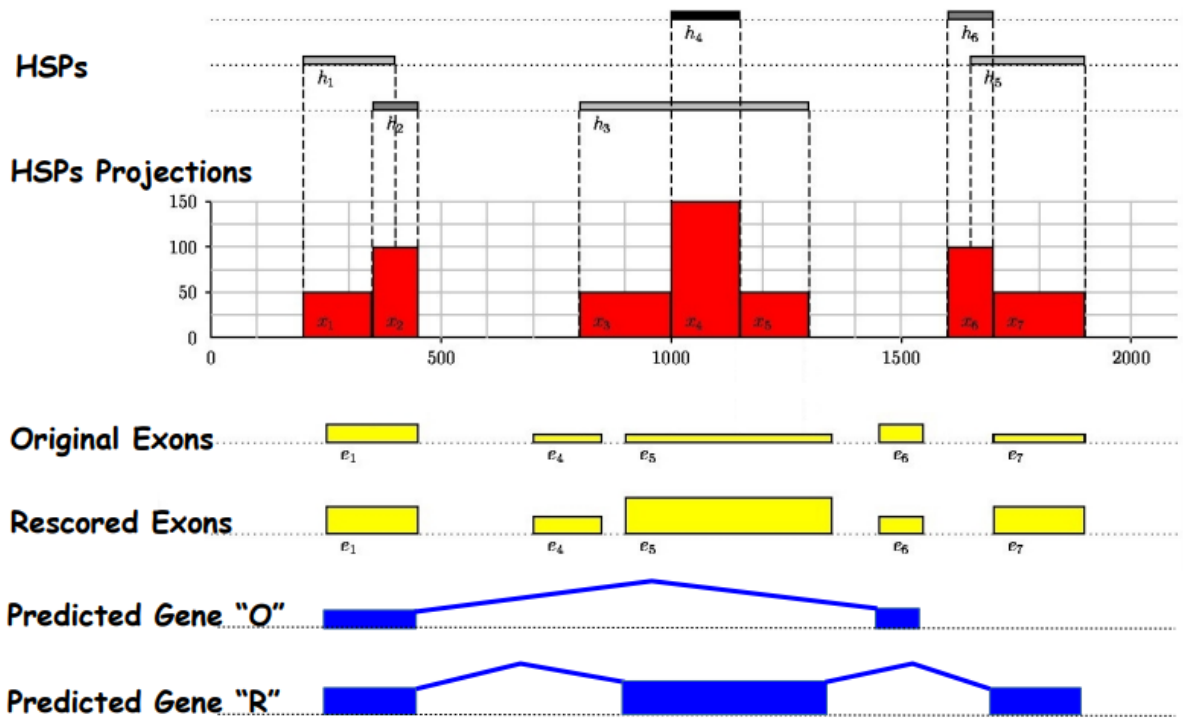
Now we will include the information about conservation:

- In the red plot we have HSP (high scoring segment pairs) from BLAST.
    - If we have a darker color, it means that we have a higher score regarding conservation.
    - There are different frames

- We can now project the HSP into similarity regions.
  Through the genome we project all the alignments from BLAST into similarity regions and now we pass the similarity regions to the program. So, exons falling in a similarity region can sum up the score of the similarity region apart from the signal scores, biases…
  So, the same exons that have been filtered, now, they have increased their score. We still have the same exons, but some of them have a higher score because they are probably more conserved.

  Now, the same algorithm as before is going to look again for the best combination of exons to find the optimal gene structure.
    - It finds a new gene structure

So, if we add homology to our prediction, we will have a more accurate result.



At the end, the gene prediction with comparative genomic evidences is an Ab Initio program that is able to re-score the exons

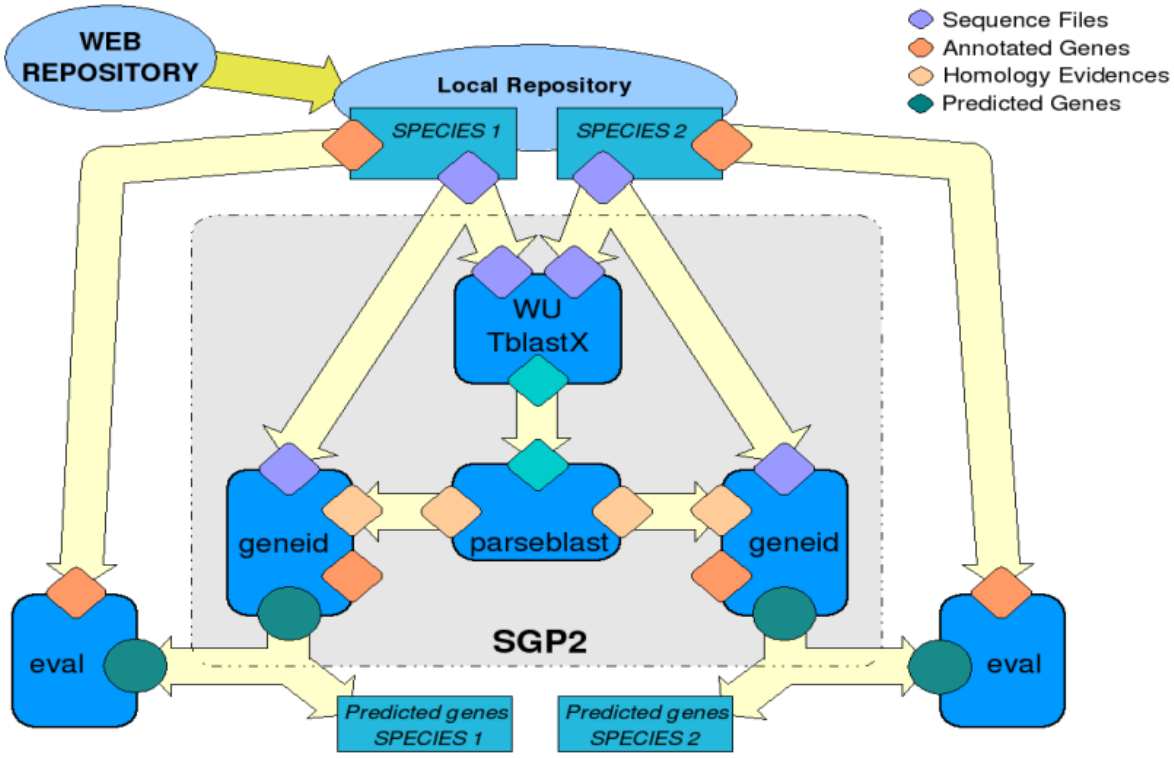
SGP2 Pipeline

Here we have an example of modular design where we are using components that have specialized functions:

- BLAST
- Geneid

SGP2 is not a gene finding tool but a pipeline (combination of different tools) that facilitates the task of integrating the alignments.
So, for a set of genomic sequences we run the TBLASTX and, as the alignments provide us coordinates for the query and the target, we can project the alignments in one species or the other.
So, having the genome of the species we can run geneid and BLAST:

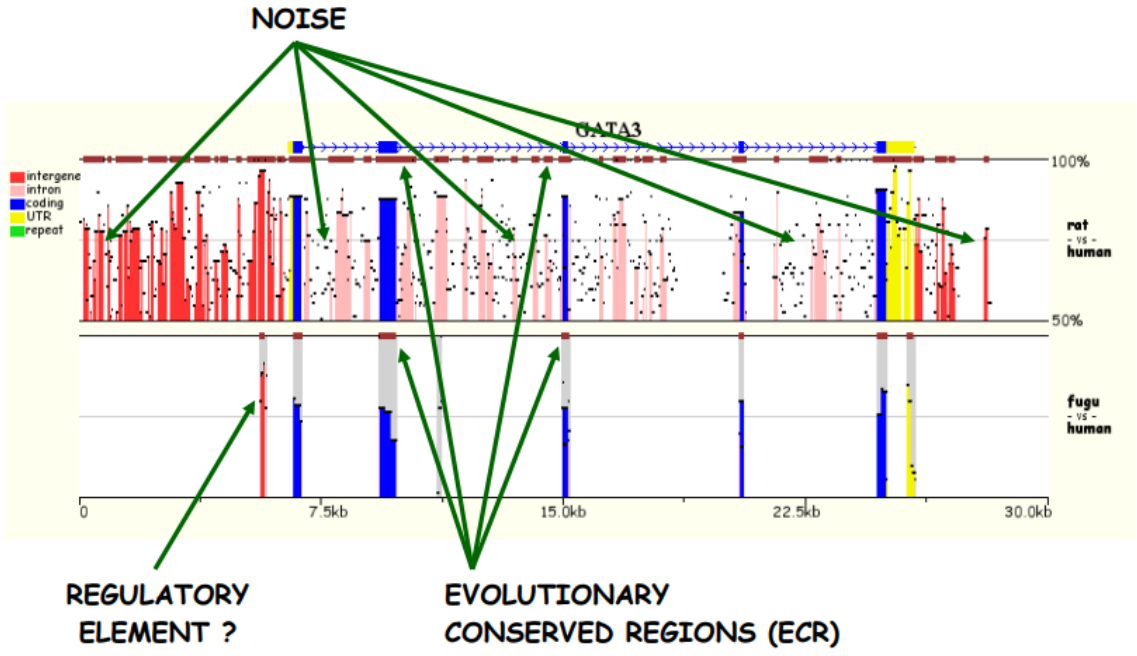
- BLAST makes the alignment of both species and the “parserBlast” can return the alignment coordinates (conserved regions) over one species or with respect to the other
- As mentioned before, geneid can take into account homology regions and score them at the exon level to predict better the gene structure
- We also have an evaluation software that allows us to compare sets of already annotated sequences with the predicted coordinates in order to find how well the prediction was. This can be used to readjust the parameters of the programs.



Phylogenetic Footprinting

Just insisting on the phylogenetic footprinting. If we compare closely related species (human and rat), we can see a clear conservation, especially in the regions that encode for a function. The problem is that we have a lot of noise.

If we use a more distant species, we can still see the conservation of functional regions. Not only proteins, but also regulatory elements. And we do not have that much noise.
So, we should find the correct distance.



So, these approaches are not only used for the gene finding tools but the promoter and regulatory elements prediction tools.

There are other approaches like “**Shadowing**”, which tries to project the changes between different sequences (instead of the conserved regions). At the end we get what is more or less conserved in all the species.
This is used to detect non coding RNAs that have some specific functions, that can be conserved or not depending on the species.

So, this method can just pick up functional elements that are not as conserved as protein sequences. For example the microRNAs that are not translated (hence they don’t have specific sequence biases as a coding sequence) but still have a function.

Is sequence conservation enough?

For coding we can find conservation but for other regions it's really difficult to find conservation (specially for those functions that may have diverged).
So, normally the programs are able to predict the main features but there are always exceptions.

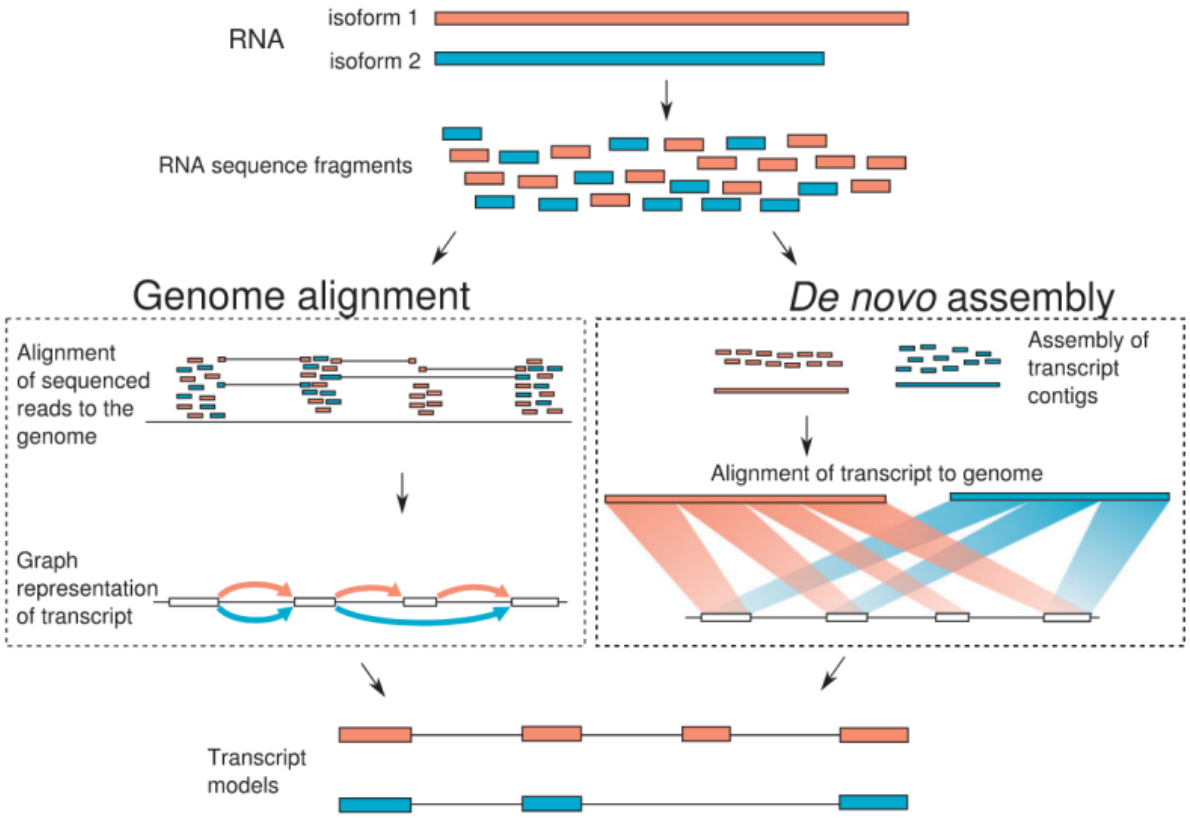
The conservation mark is not always present.

NGS & Gene Prediction

We have a set of RNAs (only coding regions) that are broken up in order to be transformed into cDNA:

- If we align this into the genome we can get the coverage that allow us to precut the exonic structure.
- We can do a De novo assembly and try to reconstruct the set of sequences for the transcriptome and then we can map the transcriptome back to the genome and find the homology regions between the assembled transcriptome and the assembly genome. This will give us the location of the exons in the genome.

At the end, using this we will probably get a representation of the gene structure of the different isoforms.
So, it does not find the optimal solution but all the possible combinations of exons that are encoded in that locus.



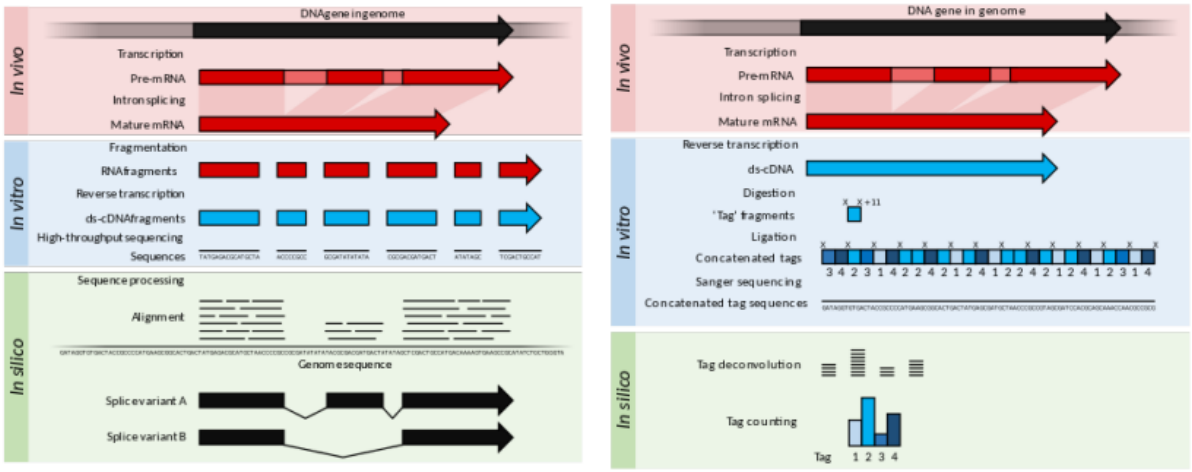
RNA-seq vs SAGE/DGE

There are some variants like RNA-seq. It tries to retrotranscrive the RNA and try to get reads that overlap on all the regions that are translated.
When we map them back we get a signal where the exons are (we can deduce introns).

We also have the Digital Gene Expression or SAGE analysis that uses a specific sequence tag. We can use an enzyme that always cuts in the same position, but the same position in different transcripts will produce a fragment with a different nucleotide composition.
Thus we will have labels for each of the transcripts.

So, for quantifying expression levels it's nice, but we are sequencing a short fragment (tag). A priori we already know the sequence of a gene and we just want to quantify it.

We can combine all the tags into fragments that can be sequenced.
So, we will get a count of all the tags and we can translate those tags into the expression level of the gene.



RNA-Seq allows us to predict the expression and the gene structure.
DGE only allows us to predict the expression.

NGS & Gene Prediction

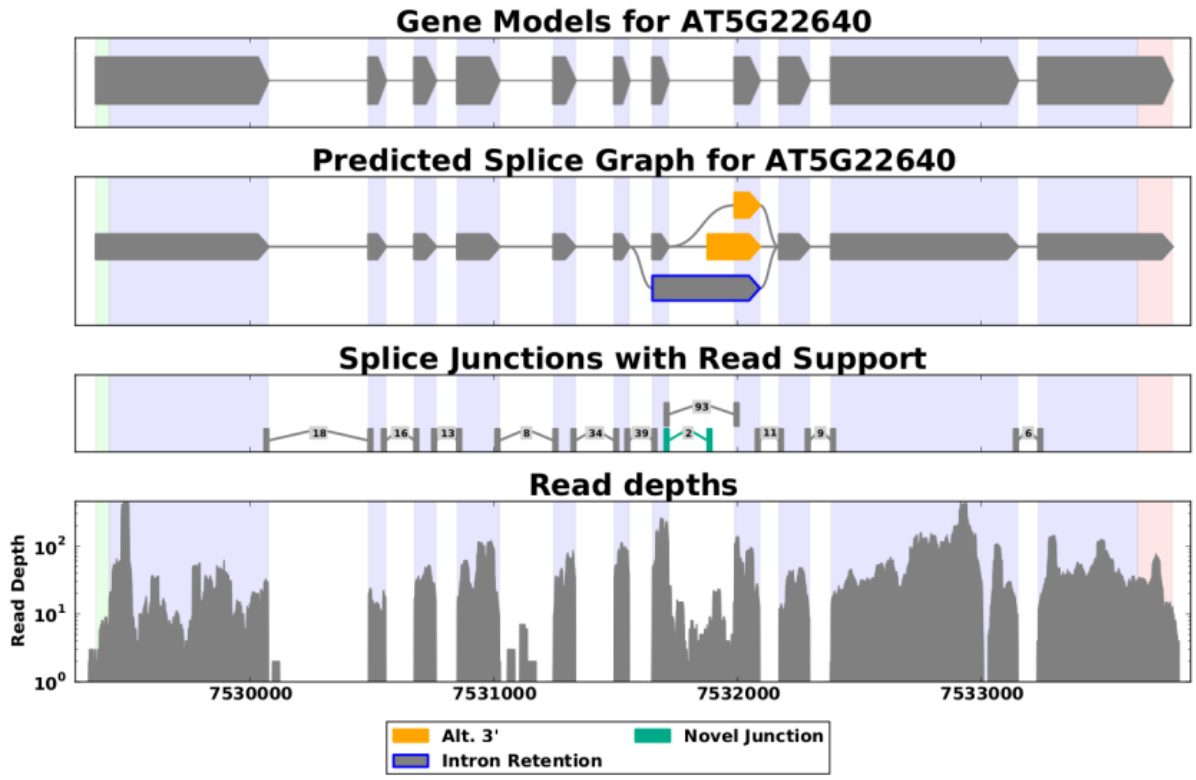
How can this be integrated into the gene finding tools?

- On top of the figure we have the standard gene model. Getting the optimal solution consists in obtaining the combination of exons that gives the best score
- On the bottom we have information taken from a RNA-seq. We have reads mapped to the genomic region, so we can see that it's not perfect (because it is not flat, there is a lot of variation). But still we can see the pattern of conservation that correlates with the exons predicted by the classical gene finder.

  However, it is supporting expression of an intron. There is not a lot of depth (coverage), but there is some.

  We also have the splice junction reads. Remember that the RNA has no introns, so when mapping into the genome we can see the splice junctions that give intronic evidence.
  Moreover, we can see that there is information about 2 different acceptor sites (gray and green). So, one of the acceptors is the canonical acceptor that was already predicted.
  By combining the information about the depth/coverage and the splice junctions, we can extend the set of predicted exons.

  How can we include this into the gene model? Instead of using this dynamic programming approach that gets the optimal combination of exons, the approach taken is making a graph of exons. The nodes are exons and edges correspond to introns



Beyond Gene-Finding

Recapitulating from the introduction, the concept of a gene has evolved from the last 100 years and, as we analyze deeper genomic sequences we find more and more functional elements that have not been taken into account before.

For example we have antisense non coding RNA that bind to the mRNA and map for degradation or inactivation (they arrest the mRNA), so they are shortening the life of the mRNA and thus it is not going to be translated in high proportions.
These are epigenetic regulatory elements and post transcriptional/post traductional regulatory elements.

More functional Elements

There are some things we need to take into account when looking for protein coding genes. We need to find the sequence, the regulatory elements to understand the dynamics of the expression of those genes, the alternative splicing…

- **Regulation:** Promoter regions, Enhancers
- **Transcription:** Alternative TSS and Poly-A sites
- **Splicing:** U2/U12 splicing machinery, alternative splicing, splicing enhancers (tells the splicing machinery which intron needs to be removed first), Cis and Trans splicing (combine exons from different chromosomes)
- **Translation:** Turnover, RNA editing (the ribosome skips the STOP codon because there is a signal in the 5’ UTR and a protein binds there), RNA interference (RNA that blocks the expression of the translation of other gene or transcript)

Pseudogenes (genes that have lost their function because a mutation has knocked out the regulatory element or splice sites or the coding region. It is still a theoretically valid gene, because it has exons, acceptor sites, donor sites… but for the cell it is not valid. Thus, they will still be predicted by a gene finding tool. The functional annotations step will try to detect them and remove them) and **parasitic genes**

Non-Coding RNAs are part of the regulatory elements (enhancers, ncRNA…) but some of them don't have a function.

There are other elements, like the **structural elements** that are conserved because they have a structural function, not because they have a coding function. Centromeric repeats or telomeric repeats are important for the stability of the chromosome or mitosis.

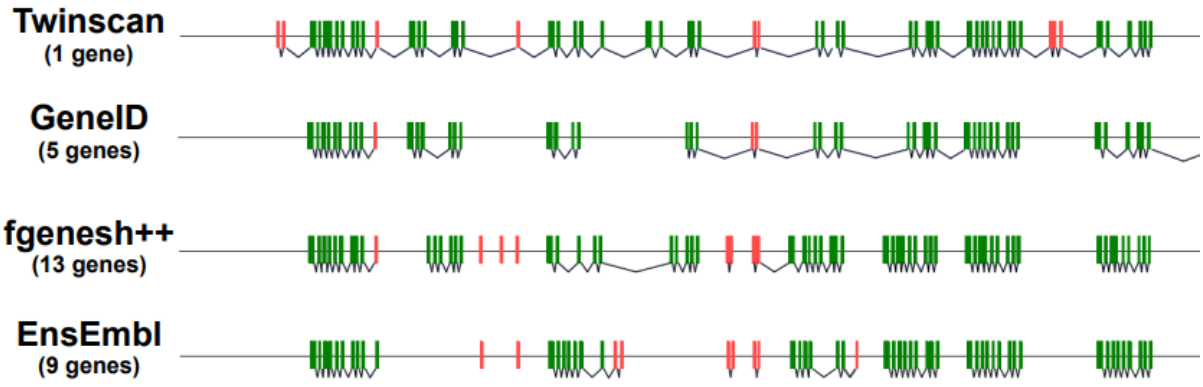
- Regulation: miRNAs and siRNAs
- RNA processing: snRNA and snoRNAs
- Protein Synthesis: tRNAs and rRNAs
- Protein Complexes: Other functional and ncRNAs (adaptors)

Gene Clusters

Other things related to accuracy (how well a genome is annotated).
Here we have an example of changes in the annotation on ensemble tracks.
So, we have a specific mouse genome region in which there is a series of genes that are duplicated in tandem.

In 2003 they already knew 4 copies of the gene:

Here we have the prediction of 4 different tools.



In most cases, they predicted correctly the real copies but also other copies.

EnsEmbl is another gene prediction pipeline that produces its own annotations. But the EnsEmbl track has a peculiarity which includes a lot of manual curations.
It has the Habana team that curates by hand.

So, the Habana team did the following annotation

So, from all the evidence from predictions, homology… they were able to annotate by hand all those regions. Improving the set of EnsEmble tracks that have missed some regions.

So, at the end there is a refinement.

10 years later, if we run again the new automated pipelines we will obtain a better result (it finds the easiest regions) but still not the correct one.
For this reason, EnsEmble decided to incorporate the Habana team annotations to fill the gaps that the programs could not find.
So, EnsEmble track is a mixture of genes that have been found using an annotation pipeline and human curated.

Note that the better the annotations, the more we will know about the function encoded, diseases…

Common Assumptions in Gene Finding

Why do we have so many problems in gene finding? Because gene tools still have some assumptions that are hard to overcome or difficult to model:

- They do not use a splicing graph. They try to find the optimal structure. This implies that they can not predict genes that overlap (even those on different strands).
- NO nested genes (genes within introns).
- NO frameshifts (RNA-editing not considered).
- NO sequencing errors (sequence quality varies from one region to another). If the genome has some misannotations, programs can not fix it. Hence, quality can be different from one genomic region to another.
- NO sequence variation. We annotate variation after annotating the genes.
- NO ambiguity codes (Y,R,N, ...).
- NO alternative splicing (only few isoforms predicted, yet when supported by evidence).
- Standard stop codons (translational recoding like in selenoproteins).
- NO split start codons (**AT**gt<intron>ag**G**tgtcg / **A**gt<intron>ag**TG**tgtcg).
- NO split stop codons (tgtcg**TA**gt<intron>ag**G** / tgtcg**T**gt<intron>ag**AG**).

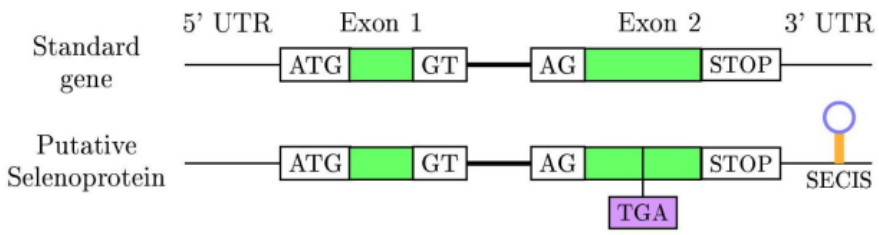
They will not be correctly predicted because it has no coding bias. But they are important because they contain the missing nucleotides for the start or stop codon.
Internal exons are easy to predict because they have codon bias.

Non-Canonical Gene Structures

Selenoproteins incorporate the 21st amino acid selenocysteine by recoding of the UGA stop codon. Gene predictors able to deal with in-frame TGA codons have shown to be very effective for this purpose.
Another example is pyrrolysine, recoded from an UAG codon. Both cases require a specific stem loop structure in 3'-UTR (SECIS and PYLIS elements respectively).

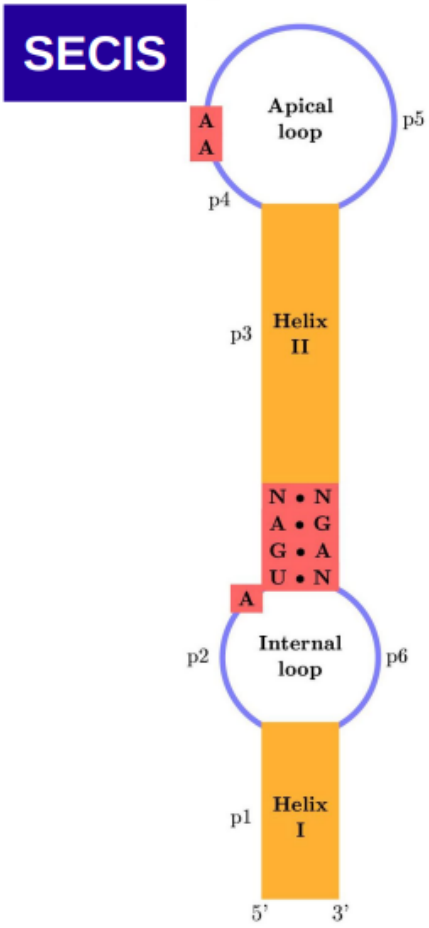
So, they have a special secondary structure in the 3' UTR that is known as the SECIS element.
To introduce selenium in the peptidic sequence, there is a need to decode the STOP codon. Expression of selenocysteine (Sec)-containing proteins requires the presence of a cis-acting mRNA structure, called selenocysteine insertion sequence (SECIS) element, which will skip the stop codon. Thus, the sequence will be extended until the next stop codon.

We use a special gene finding tool to detect this type of non-canonical genes.



We can build a model that detects which UTRs can have the SECIS element:

- Identify Helix I, internal loop, quartet, Helix II, apical loop, Helix II, quartet, internal loop, Helix I.



When comparing genomes, we can see that in some cases there is still a conservation of some nucleotides after a STOP codon.
This could be because it encodes a selenoprotein and thus we have to take into consideration the next stop codon that we find downstream.

Non-coding RNAs constitute a vast array of genes that do not necessarily encode for proteins. In consequence, while standard gene prediction programs cannot detect them using codon bias sensors, it is possible to identify them with a combination of RNA-Seq data and a different DNA composition that includes in-frame stop codons.

Pseudogenes are derived from functional genes through retrotransposition or duplication but have lost the original functions of their parental genes. Their high degree of evolutionary conservation and similarity to functional genes are significant enough to confound conventional gene prediction approaches.

Fusion genes combine part or all of the exons from transcripts of two collinear gene loci. Those fusion genes are often the result of a structural variation at chromosomal level, such as a translocation, an interstitial deletion or an inversion. It has been shown that tandem gene pairs may also contribute to increase the protein catalog in eukaryota.

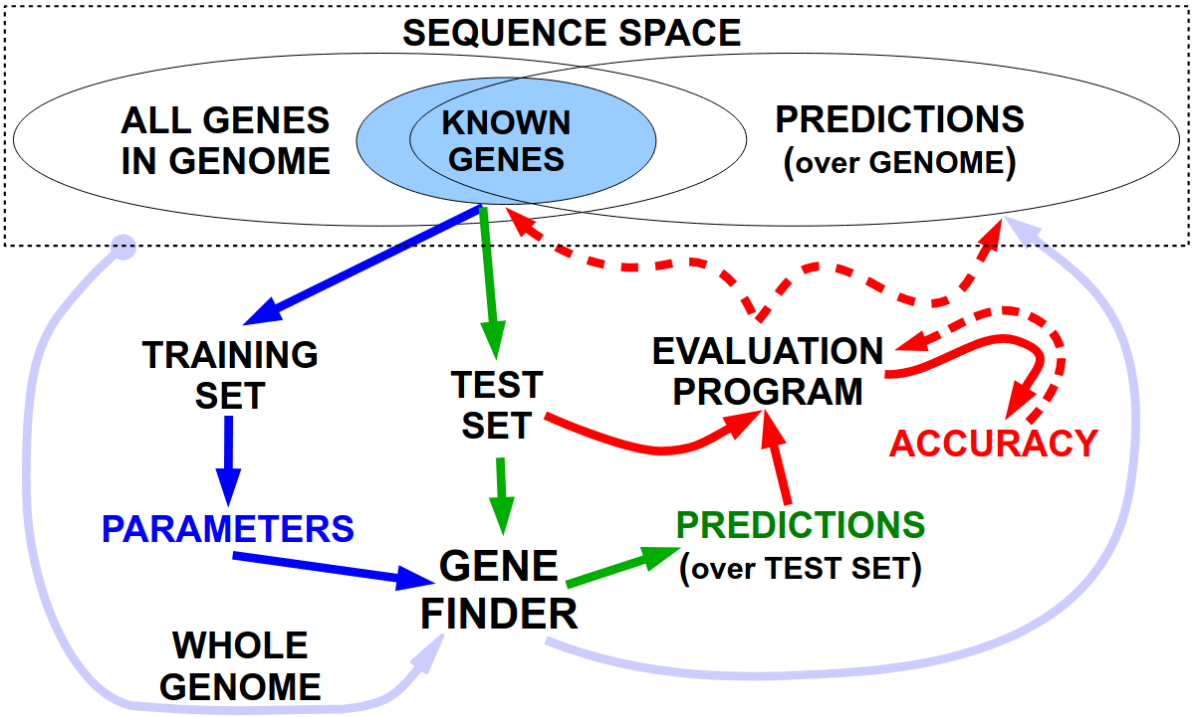
Trans-splicing is a transcriptional process where initial exons from one transcript are merged with exons from another transcript, far downstream or even located in another chromosome sequence. For instance, up to 70% of Caenorhabditis elegans mRNAs begin with the spliced leader sequence (SL, 22bp), which is not associated with the gene

10th Theoretical Class

Unit 6. Gene Finding Assessment

We have to define how to perform the assessment and have to determine a standard against which we will compare the prediction. Usually this is the set of known genes, but we have to take into account that the databases and the annotations are not complete or good enough. It is not the same to do the assessment on a human genome or fly genome (very studied and complete), than doing it on a newly discovered or obscure species. So, we need good evidence that the genes are really there.

Often we split the set of known genes into 2 datasets. Training set, to set the parameters of the program, and test set. We know the coordinates of these genes, so using the parameters we have trained on the gene finder we can produce a set of predicted coordinates.

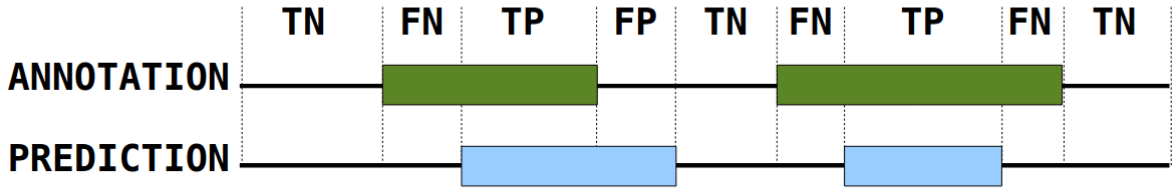


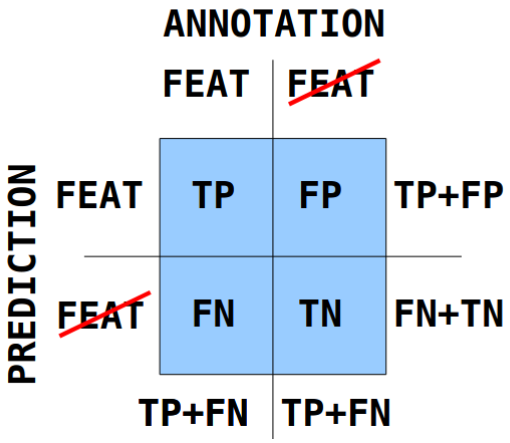
As now we know the true and predicted coordinates, we can compare the two. The program gives us a measurement of accuracy, in an estimated percentage. This means that a-priori we don't know exactly which genes were correctly predicted and wrongly predicted.
Now that the gene finding tool has been trained with the parameters, with a certain percentage of accuracy on the test set, we can make predictions finally. The better the program works, the more overlap we will have among the set of predicted genes and the real genes.
Keep in mind that, if the set of annotated genes is biased for some reason, the gene finding tool will also be biased and the measure of accuracy will be overestimated. Also keep in mind that we are training the program on a subset of the genome, so if we get 75% accuracy for a little subset of the genome, we will assume that the accuracy of the whole genome will also be 75%, which is an estimation.
We can also reassemble the genomes many times with different sets to close the gaps or change some parameters to see different results. Also if the annotation has new versions we are able to redo the assembly.

It is important to differentiate between the annotation tracks (curated by somebody, ‘real’ genes) and the prediction tracks (what we have predicted with software). If we have different programs, we may have different prediction tracks because they use different algorithms or parameters. The annotation track is usually a combination of many prediction tracks that have been demonstrated experimentally and curated by someone, an annotation track will always have more accuracy than a single prediction track.

These are some of the accuracy measures we can use at the nucleotide level. We have the sequences represented as a line and their corresponding exons. So at the nucleotide level we are just looking for true positives (nucleotides that match in both sequences). Those two measures in the formulas are sensitivity (recall) and specificity (precision). We avoid the true negatives because we cannot truly know them.

Accuracy Measures [Nucleotide level]



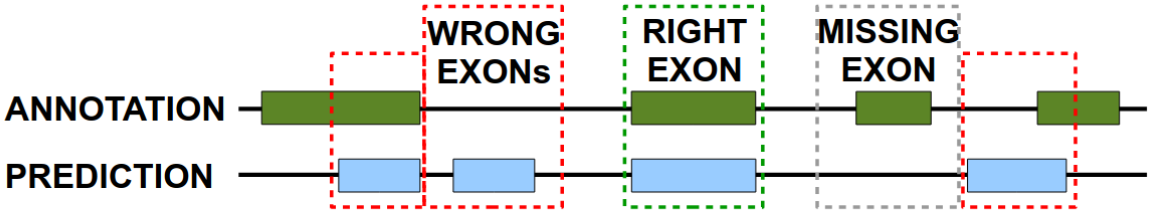


$$SN_{nucl} = \frac{TP}{TP+FN} \sim \frac{Features\ FOUND}{Features\ ANNOTATED}$$ RECALL

$$SP_{nucl} = \frac{TP}{TP+FP} \sim \frac{Features\ FOUND}{Features\ PREDICTED}$$ PRECISION

Now we can also look at the exon level. So now we are looking at the starting and ending positions of the exons, as well as the nucleotides matching.

Accuracy Measures [Exon level]



$$SN_{exon} = \frac{RIGHT\ EXONs}{ANNOTATED\ EXONs}$$

$$SP_{exon} = \frac{RIGHT\ EXONs}{PREDICTED\ EXONs}$$

Assessment at different levels:
nucleotide, exon, transcript, gene, etc...

Other measures:
Average Correlation (AC)
Correlation Coefficient (CC)
Missing/Wrong Exons (ME,WE)
Missing/Wrong Genes (MG,WG)
Split or Joined Genes
and so on...

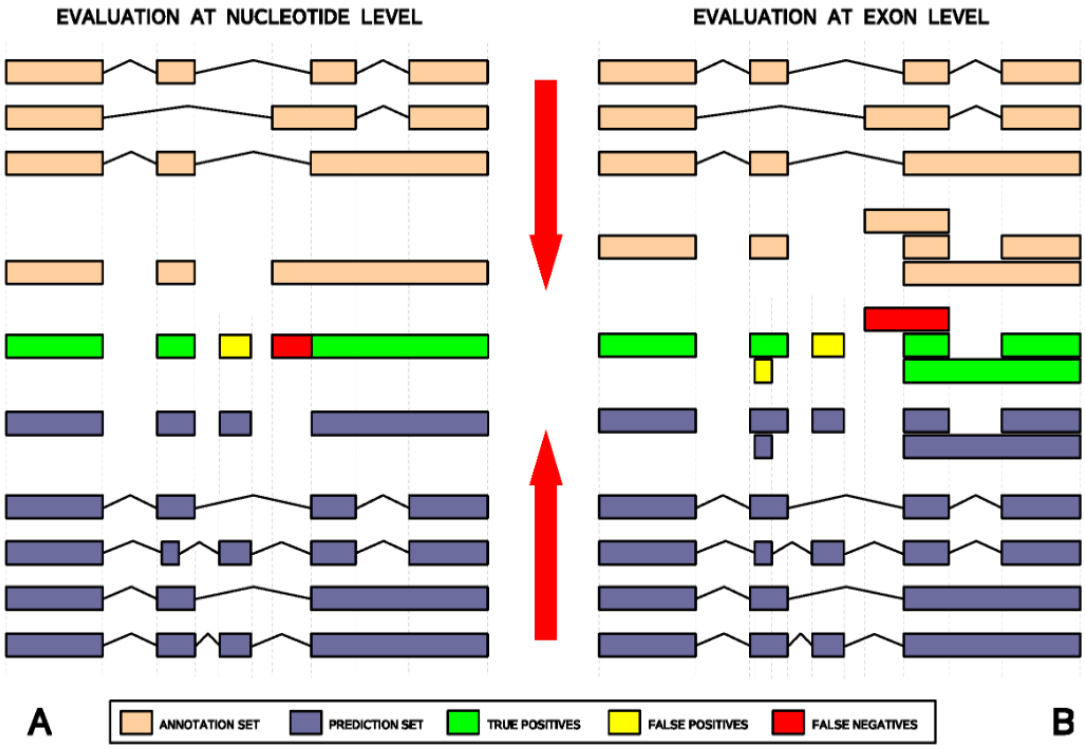
For an exon to be true positive, the start and end coordinates must match, as well as being true positive in nucleotide level. We also have measures of sensitivity and specificity. These measurements will always be lower than just at nucleotide level.

If we extrapolate this to transcripts, the right transcript will be the one that has the same exon structure as the annotated one.

We also have to take into account when we go to a higher level, from exons to genes, if the program is predicting 2 separate genes instead of one or joining 2 separate genes to make one long gene. So apart from the ratios of accuracy described above we can also take the ratio of splits of some genes. For example, something that is a gene with 5 exons can be predicted as 2 genes, one with 3 exons and another with 2. If this happens often, we have a high ratio of splits.
These measurements of accuracy evaluation can help us determine which set of parameters is best for our prediction, since we would want to maximize the accuracy.

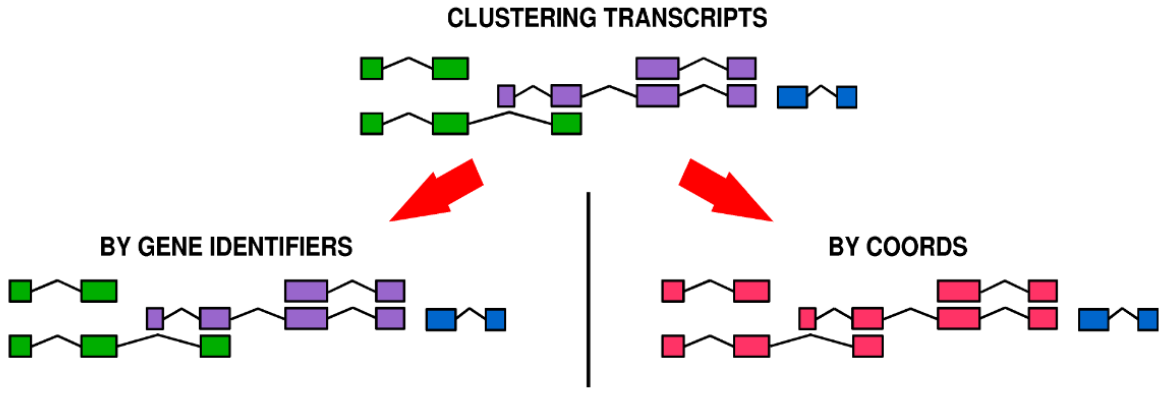
Here is another way to illustrate what we were doing:

Measures Illustrated [1]



Now the problem comes with the genes, because the structure of one gene may overlap with the structure of another. It can happen that we have 2 genes that share an exon, so we have to define if it is a single locus or 2 loci. To differentiate, we may use the annotation of the genes (gene identifiers), or their coordinates, as illustrated below:

Measures Illustrated [2]



The simple choice is using coordinates: just merge everything into a single set of overlapping coordinates.

How well do you do?

There are some competitions where you can take your program and compare its accuracy against other programs in an unbiased way (same data and conditions):

The Genome Annotation Assessment Project (GASP) was launched in 1999 to evaluate the performance of current computational gene prediction programs on the well-characterized Adh gene neighborhood region, 2.4Mbp, of the Drosophila melanogaster genome. These kinds of evaluations were inspired by the structural biology Critical Assessment of Structure Prediction [CASP; Lattman, 1995], which became a regular procedure to monitor state-of-the-art methods to model protein structure [Moult, 2005].

The ENCODE GASP (EGASP) community experiment, within the context of the ENCODE project, developed in 2005 a more exhaustive evaluation procedure on the GENCODE reference set of the human genome spanning 30Mbp over 44 selected sequences.

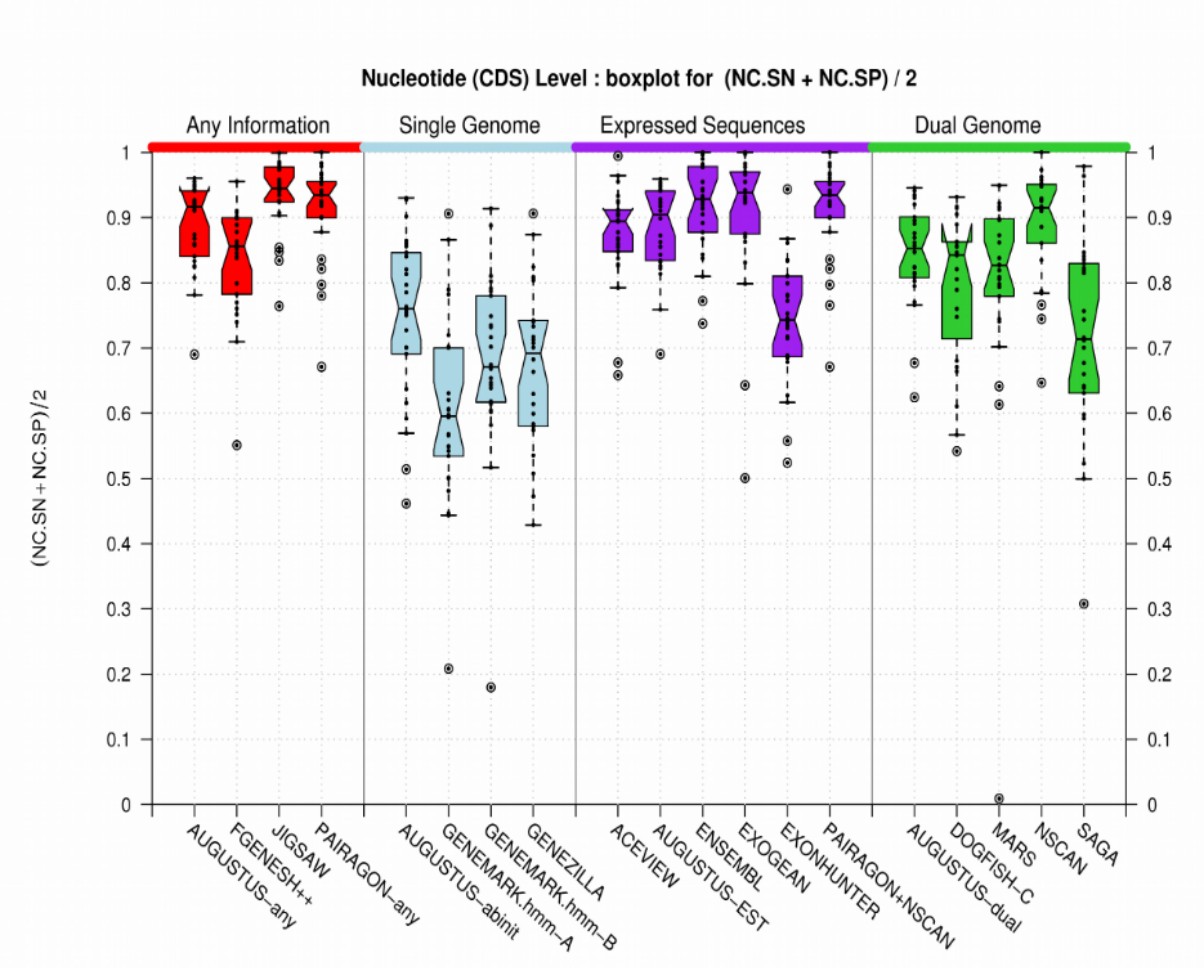
Later on, in 2008, the nGASP Consortium conducted the nematode Genome Annotation Assessment Project (nGASP), focusing the interest in the annotation of protein-coding regions from Caenorhabditis elegans, four additional Caenorhabditis genomes and other nematodes.

More recently, the RNAseq genome annotation assessment project (RGASP) evaluated the current progress of automatic gene building using RNAseq data to determine alternative transcript isoforms and quantify also their abundance.

The rules of the competition are similar in all GASPs: 1) access to a training set of exons is granted to participants in both cases, and 2) posterior submission of predictions is compared to expert annotations to calculate the accuracy of each system. It was established in all these studies that while a significant fraction of the coding sequence was successfully characterized at nucleotide level, the correct intron/exon structures and alternative splicing events were captured only in about half of genes which suggests substantial room for improvement

The EGASP gave us insight into the nucleotide level across different programs:

NUCLEOTIDE LEVEL



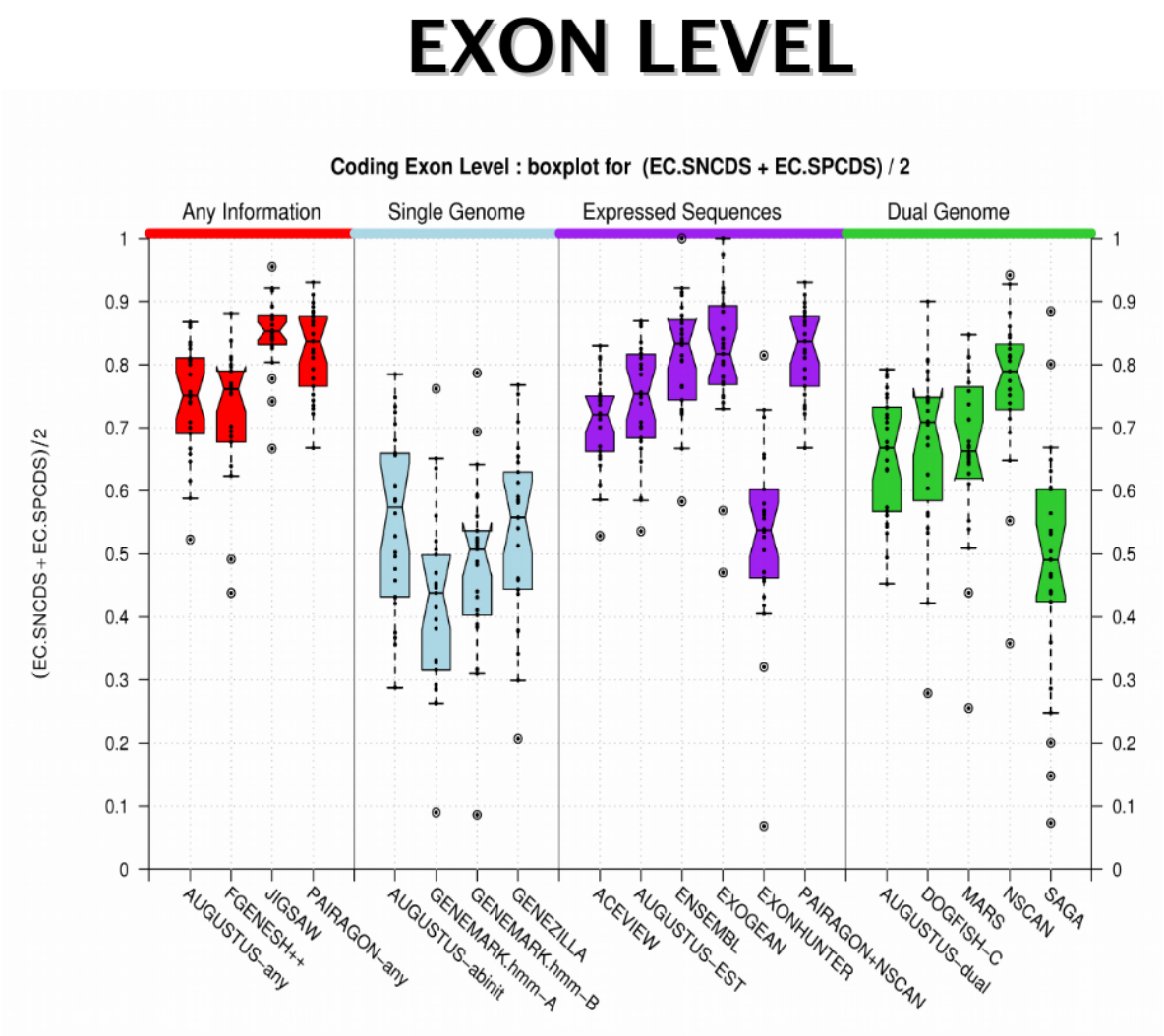
We are plotting the average sensitivity and specificity, and every dot is the result of one of the sequences for each program. We should expect that, if the program is stable (is not affected by the density of the genes or any feature on the sequence), it should behave the same in every sequence. If there is a bias because of the sequence content, composition, similarity, etc. we will have a wide distribution. The more variance the result has, the less stable the program is.

There are always some sequences that are super low with respect to the boxplots, and these are likely the Y chromosome because it's very small and contains little genes.

Notice that we are grouping by categories (look at the colors of the boxes). The light blue ones are ab-initio, so we are considering the sequences and the models for donor sites, acceptor sites and sequence biases. They perform on average worse. Those that are based on databases or comparative genomics approaches (purple=homology and green=comparing 2 genomes) perform on average better than the ab-initio programs.

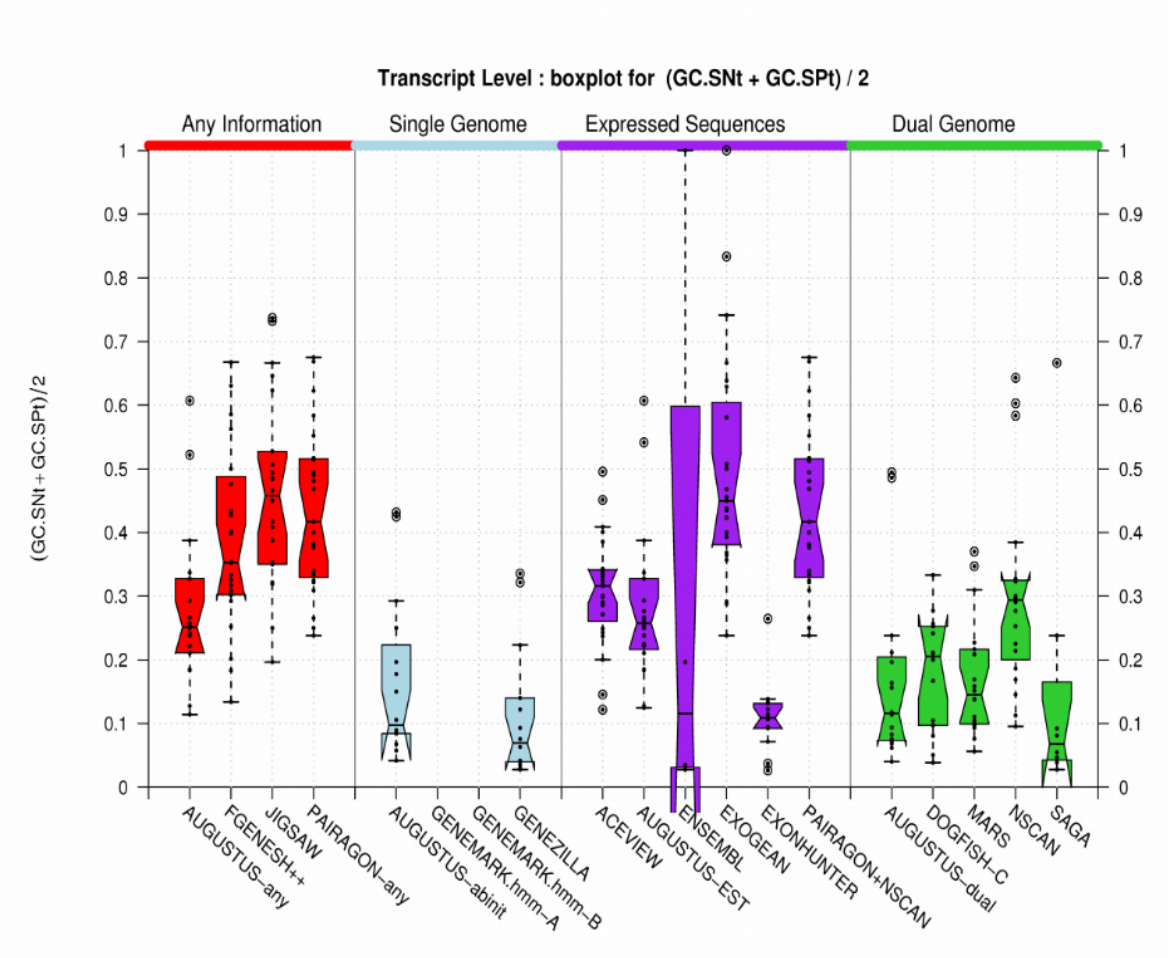
The red ones take into account any information as possible (homology, conservation against other species, RNA-seq data…) and they perform the best. The problem is that depending on the species we want to annotate, we can have a problem because maybe we lack annotations. For those species, we can only run ab-initio tools (avg 75% accuracy if we are lucky). The point is, at nucleotide level, the more information we have the better accuracy.

Also, we can look at the exon level predictions:



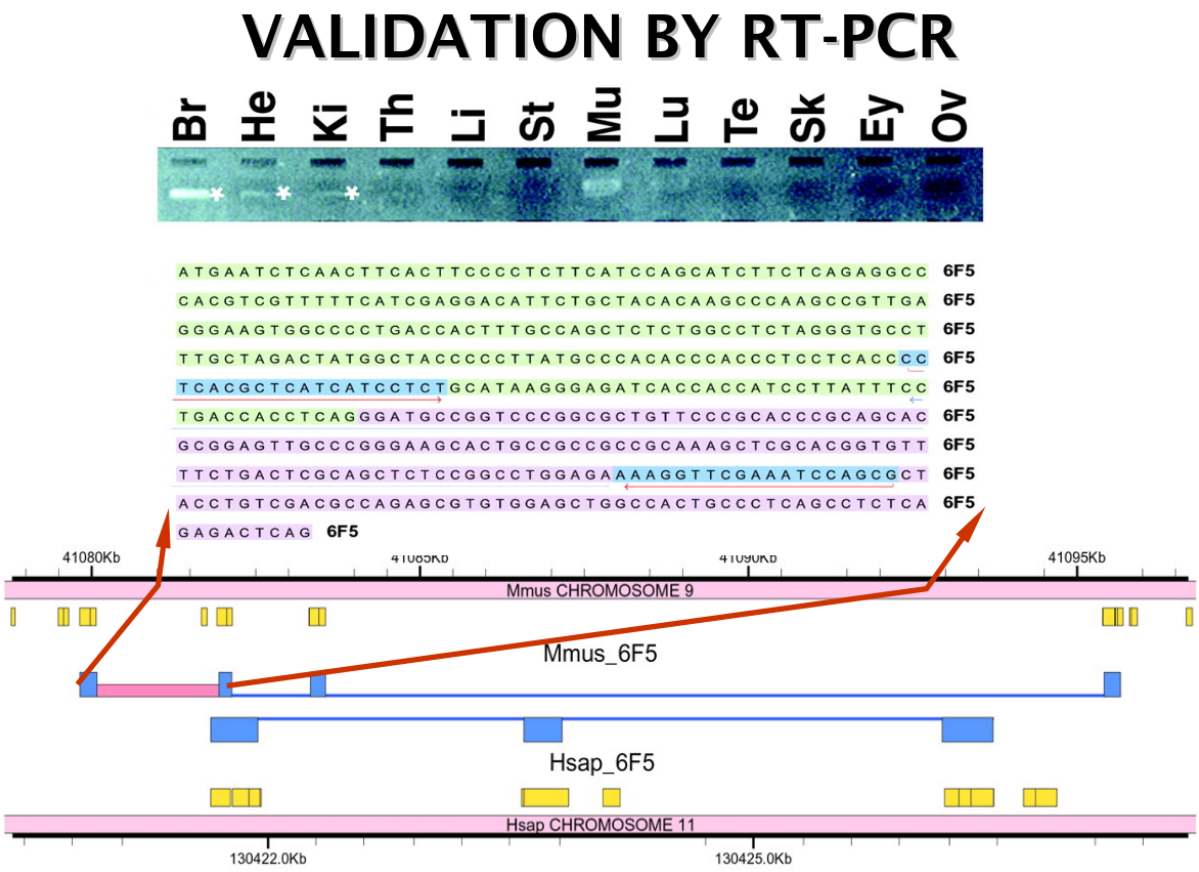
We expect the quality to be overall less, and we can see that clearly in the plot. The best performing tools now have 75% or so accuracy on average. If we go to transcript level…

TRANSCRIPT LEVEL



The accuracy is super horrible as expected, because we are being very strict (for a transcript to be a true positive, we have to annotate all the exons of that transcript).

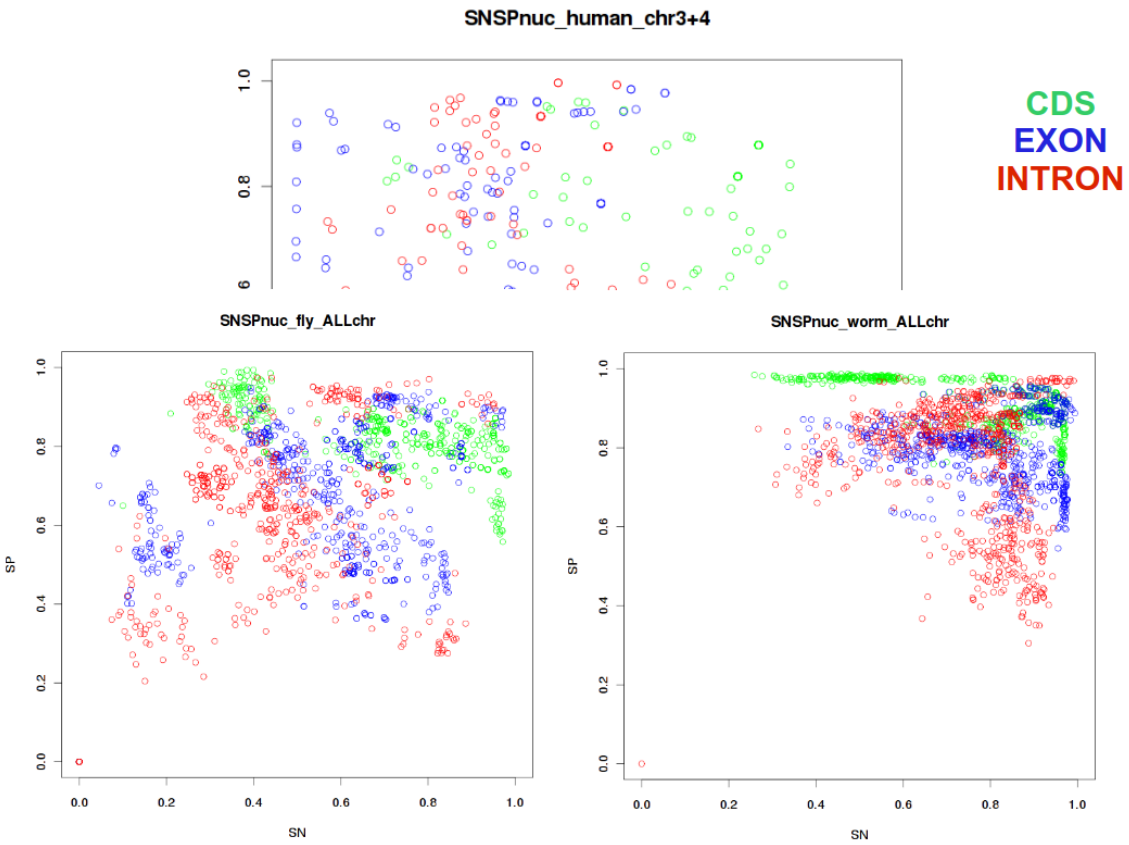
If we have predicted structures across human and mouse and had conservation, they were able to validate them experimentally to confirm if it was a false positive or not.
In the old times, they validated structures by designing primers based on the sequence of the chromosome and the coordinates of the exons. They did a PCR with primers designed on the 5' exons and the 3' exons. No entenc del tot aixo (00:47:12 fins a 00:52:46)



Nowadays, we don’t need this because of RNA-seq data.

This is related to the RGASP project. They took different species with different densities of genes, tested different sequencing techniques and did evaluations at nucleotide levels.

Evaluation @ Nucleotide Level

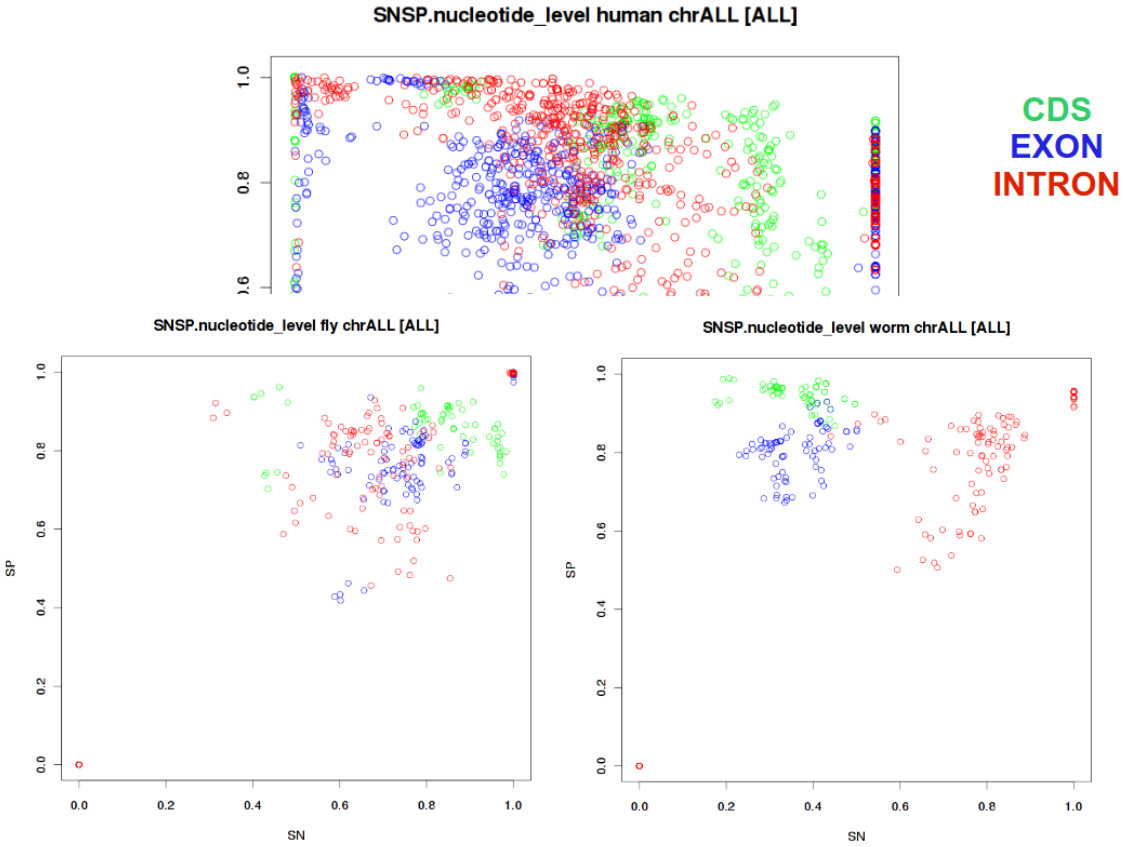


We see in different colors the predicted CDS exons and introns.
Every point is the relationship between sn and sp.
The more up and to the right of the plot, the better quality. We can see the coding sequences (green) get on average better scores than the exons (blue) due to the coding bias (the UTR exons are not as well predicted as the coding ones due to the coding bias).

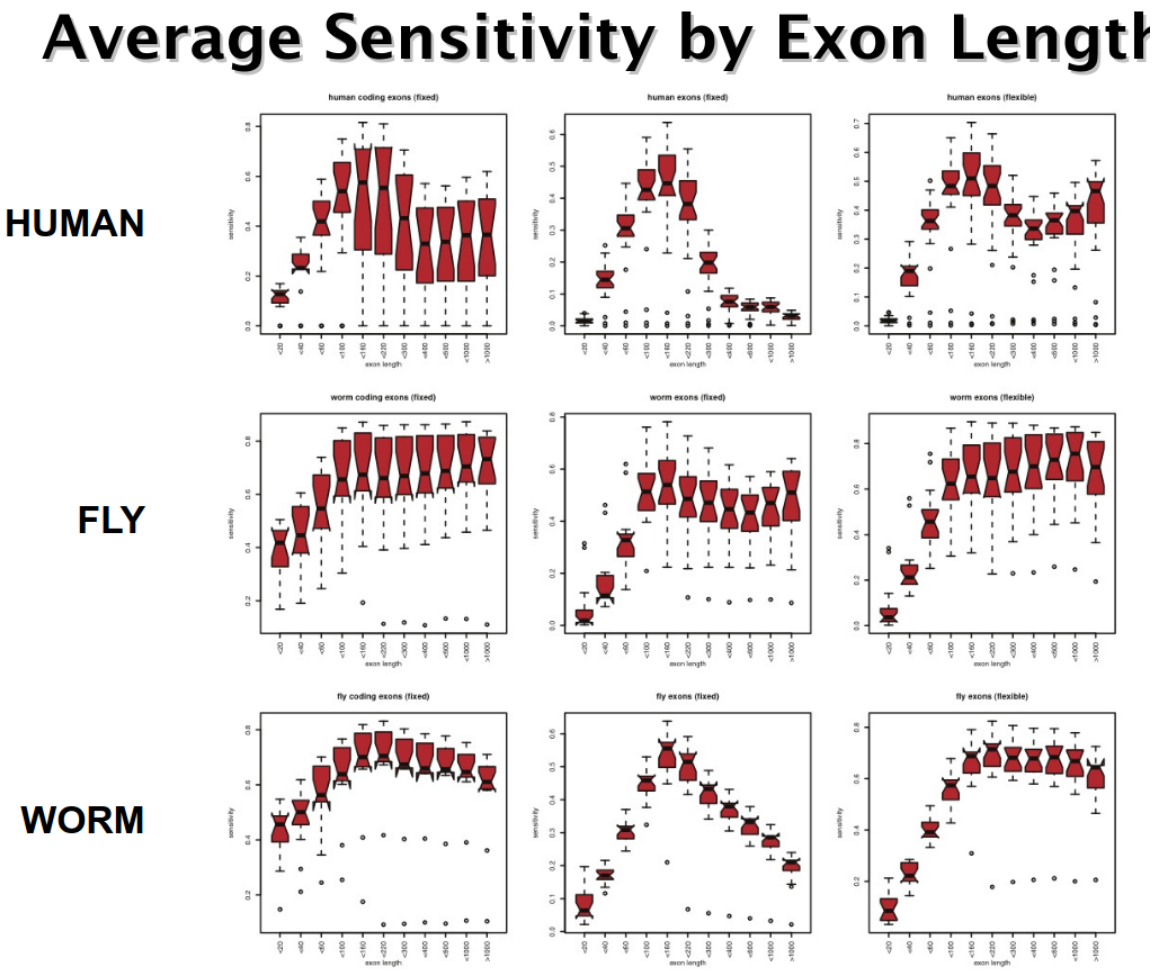
- The coding exons are the ones that are better predicted (CDS)
- Exons can contain UTRs that are not that well predicted

The C. elegans has a very compact and simpler gene structure so the CDS are extremely easy to predict. On humans we have a lot of spread of data and we would like to improve the quality of all of them.
Let's see another evaluation at nucleotide level:

Evaluation 1.2 @ Nucleotide Level



There is a bias. Let's see it here.



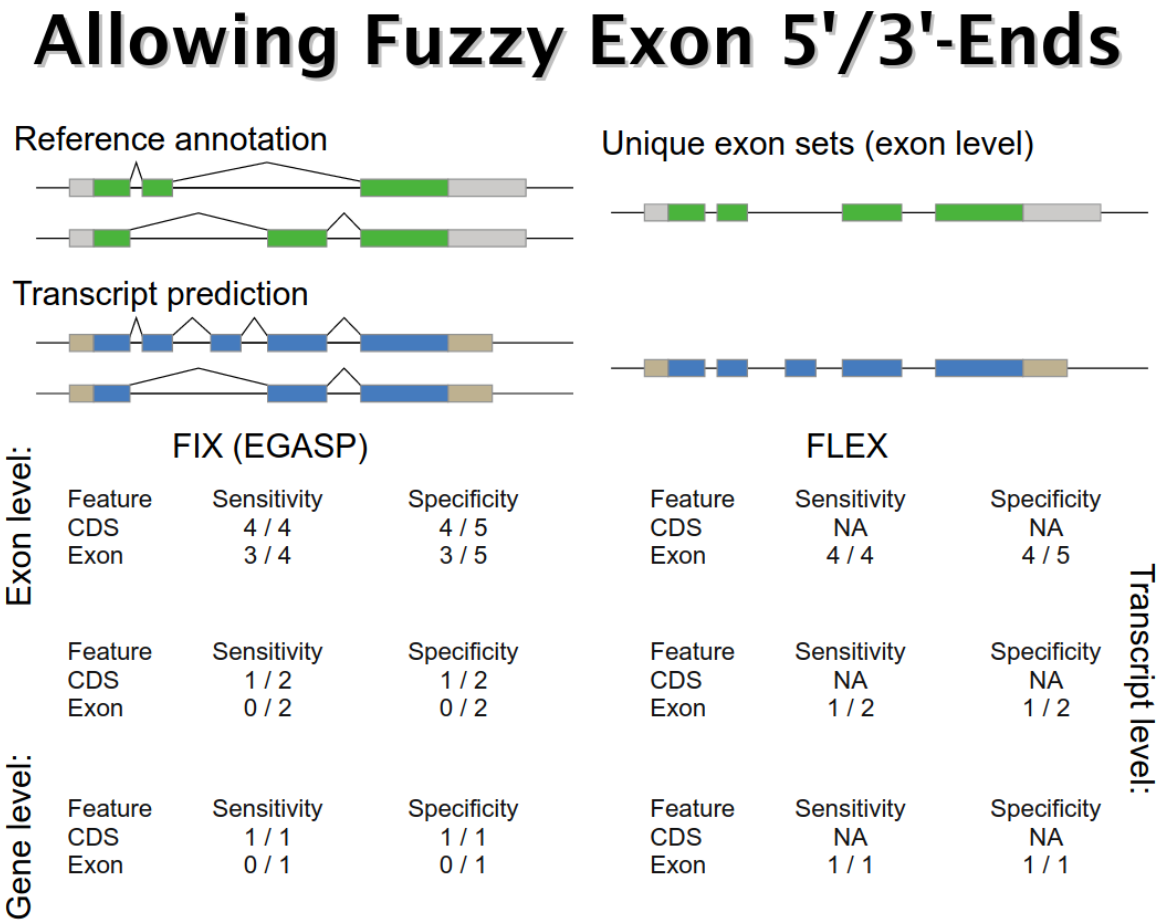
Exon length vs sensitivity

The short exons are worse predicted than others. On average we have better scores on a simpler genome (worm aka C. elegans) than complex genomes, but we still have the effect that short exons are predicted to be the worst.
Also, remember that if you want to make predictions at transcript level you need information on the UTR exons, so if we are seeing a bad score for exons this will be a problem even if we have a good score for CDS.

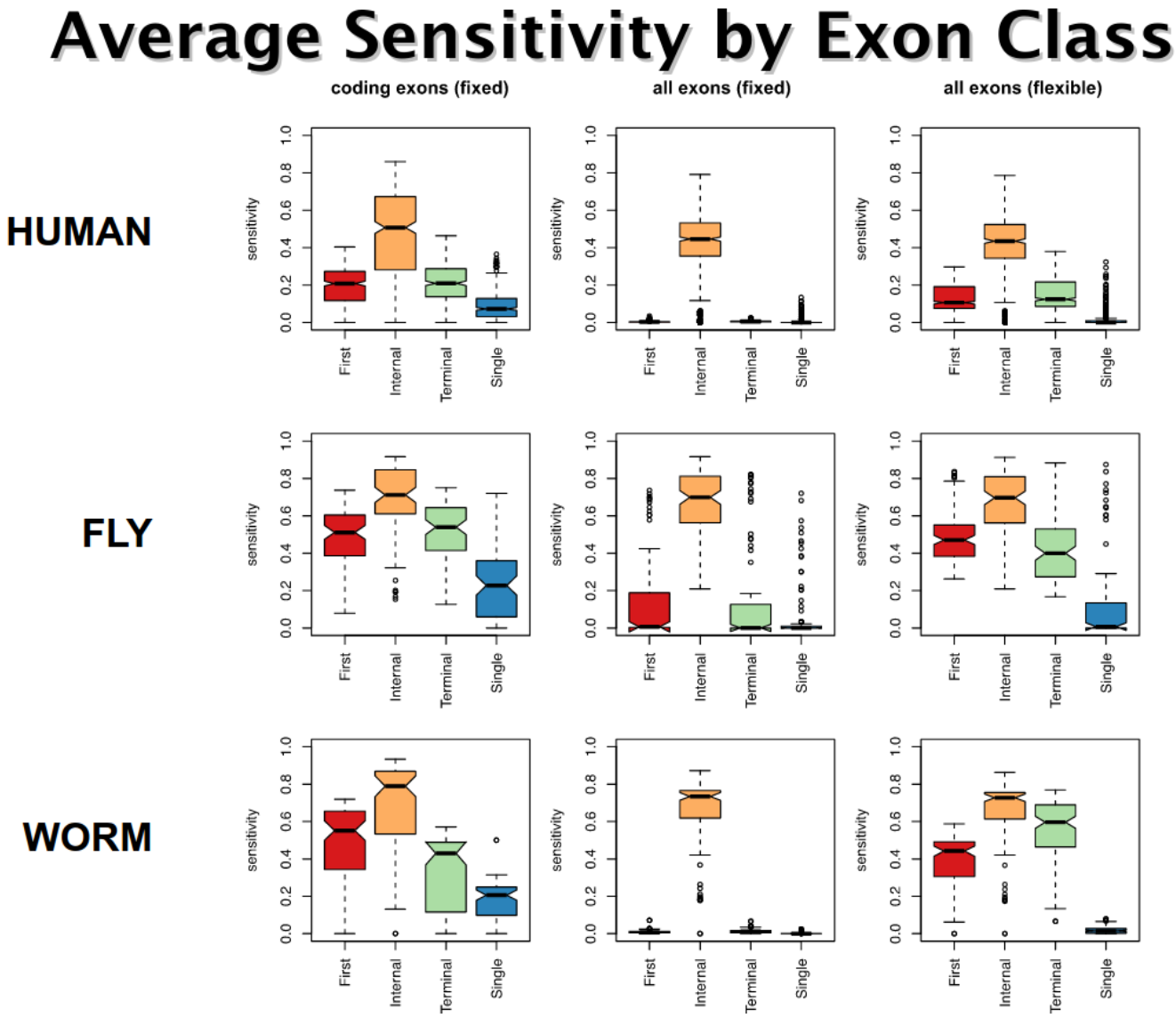
On EGASP, we are assessing transcripts at two levels:

- fixed: forcing that all exons and coordinates are the same
- flexible: considering only the unique exons and focusing on the coding (the UTRs were not considered). In principle, this improves the prediction

See in the next image the difference.



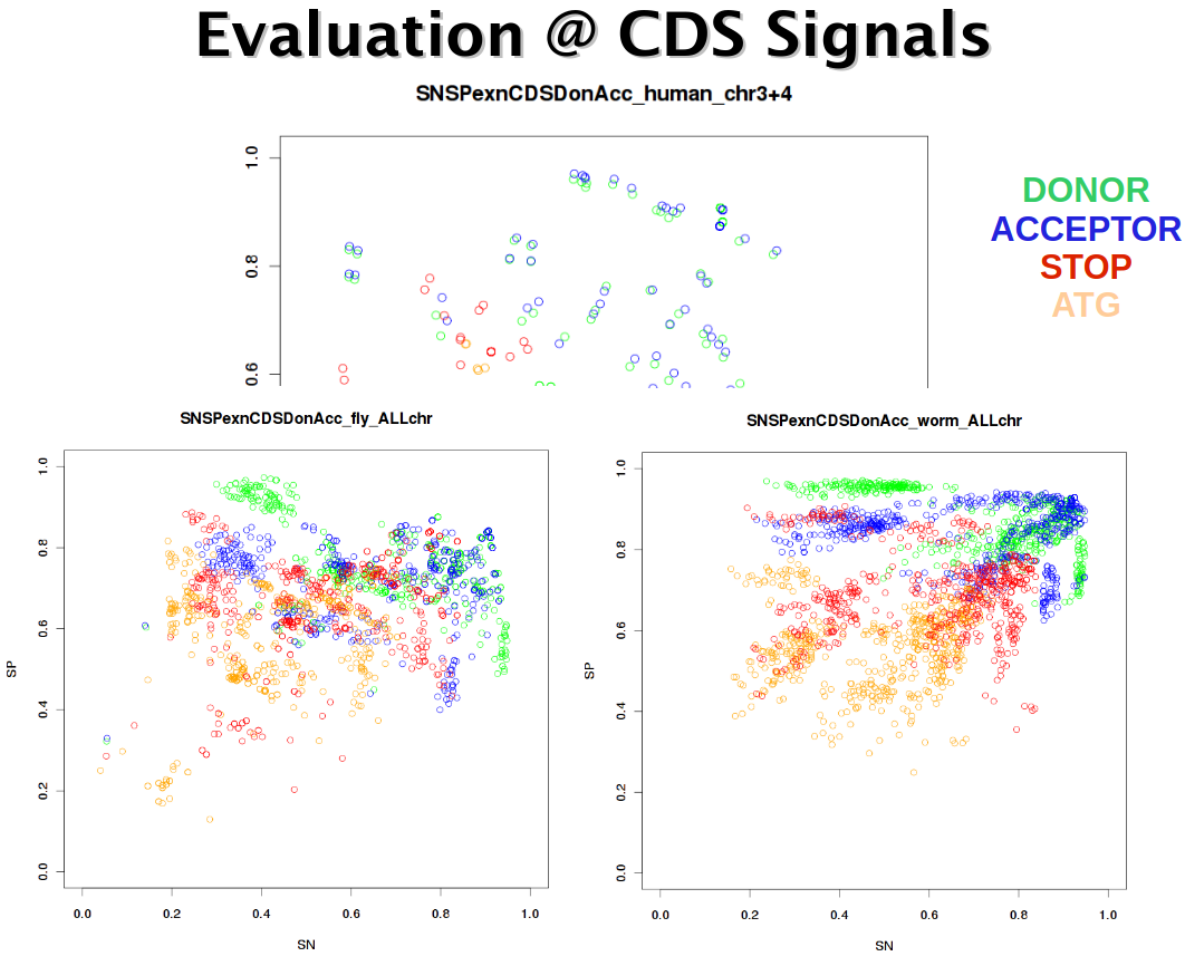
If we look for the accuracy by exon type we see this:



We are looking at single, internal, initial or terminal exons.

- Initial exons start with ATG and has a donor site
- Internal has an acceptor and ends with a donor site
- Terminal start with acceptor and has a stop codon

We see that internal coding exons are better predicted than any other internal exon. This is true also for all exons. This happens because of the signals: internal exons have an acceptor and a donor, and those are strong signals with respect to other signals. This is demonstrated in this plot:



Acceptors and donors get better scores than other signals. This happens in all species, not only humans. So if my exon has a donor and an acceptor, probably it has better score so it's better predicted.

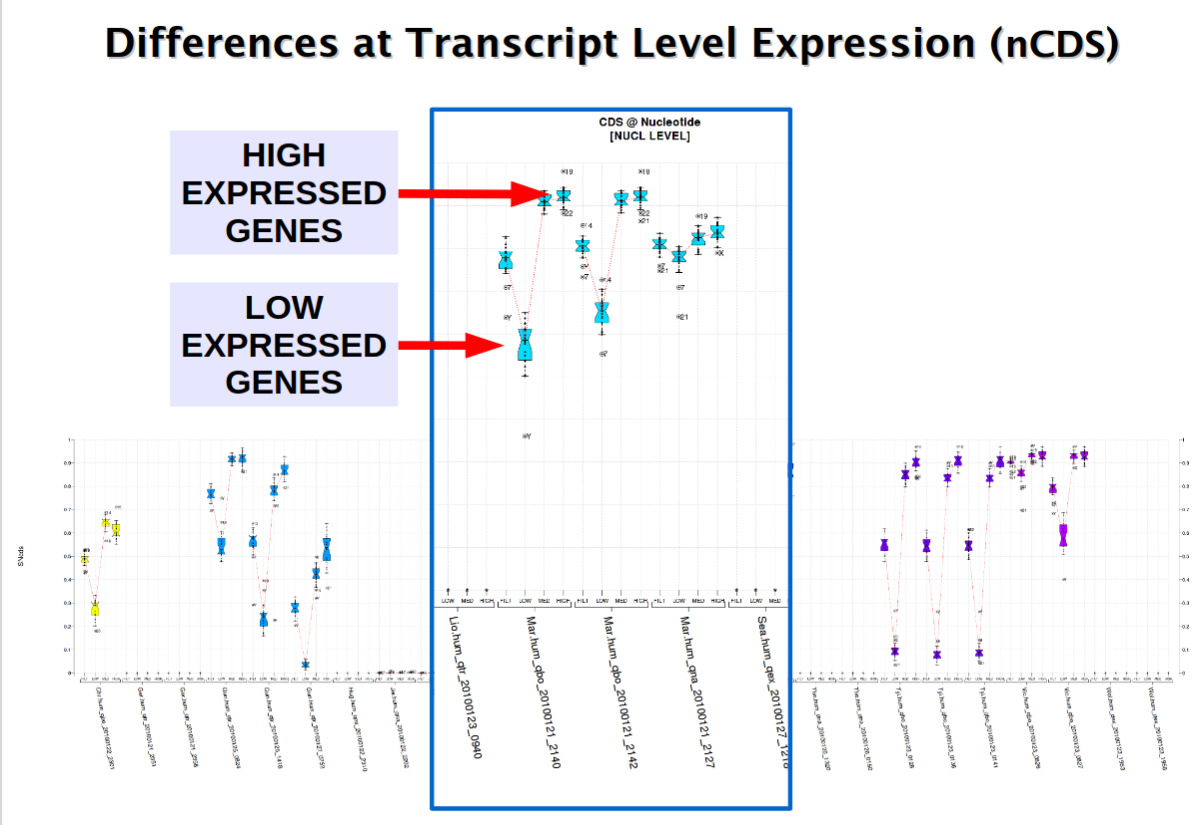
The programs could also use RNA-seq data to evaluate the expression levels of the genes. This is important because there can be a bias depending on the expression level. Are highly expressed genes better predicted than low expressed genes? In other words, is the training set biased? Is it enriched in any kind of genes? Maybe we only have housekeeping genes in the training set because they are more abundant, so we could only predict genes that have similar structures to them (only a few).

Defining Expression Level Sets

Low RPKM < 1
Medium 1 > RPKM < 10
High RPKM > 10 (pseudogenes treated separately)

Organism	Low	Medium	High	Total
Human	9875 (34%)	7696 (27%)	2606 (9%)	20177 (70%) / 29046
Human Pseudogenes	4188 (36%)	838 (7%)	0 (0%)	5026 (43%) / 11784
Worm	5855 (29%)	6422 (32%)	5516 (27%)	17793 (88%) / 20158
Fly	1630 (13%)	5705 (47%)	4756 (39%)	12091 (99%) / 12240

Since they had data from RNA-seq, they split the assessment into expression levels, so now for every program we have boxplots split for every expression level:



For each program we see that there is not only 1 boxplot but it is actually split into different expression levels (low, med, high). This happens at nucleotide level and transcript level. So there is clearly a bias because of the data in the training set (they are overrepresented).

(summary en 01:02:08) So to summarize, maybe the programs are overtrained for what we know and what we know is already biased (lack of genes in certain categories) but with RNA-seq data this can be overcome. Another problem is that the models are too simple, so we have to include something else into the model to make it work (maybe the loci are more complex than we expect). Also take into account that nothing is perfectly annotated.

After Match (*GASP)

- Are programs over-trained on currently available whole genome annotations, which makes hard to integrate new evidences into novel transcripts ?
- Are current models still too simple in comparison to the growing complexity underlying loci functional features ? … specially on human genome...
- Remaining issues on gene-finding:
  - Given current scale of analysis, a common notation is required for a proper comparison of features (gold standards).
  - UTR exons prediction.
  - Defining proper start/stop signals.
  - Single CDS exon genes.

- Long-read RNA-seq Genome Annotation Assessment Project

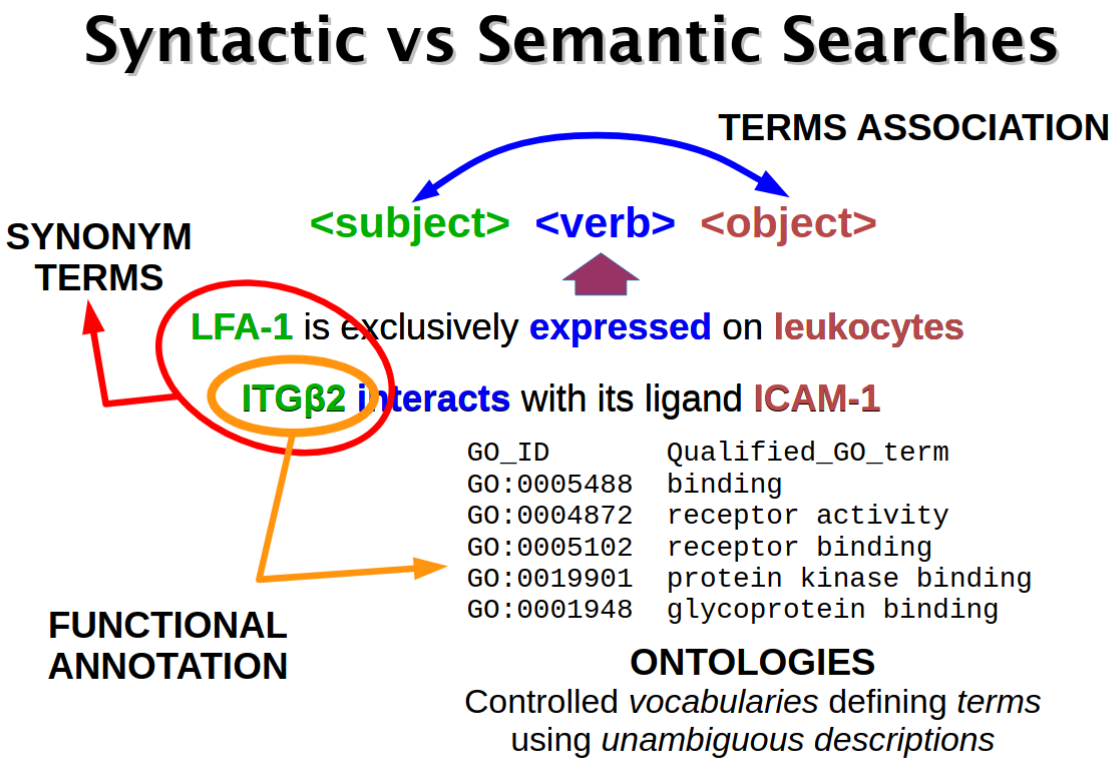
https://www.gencodegenes.org/pages/LRGASP/

Unit 7. Functional Annotation

How do we search data in a database? We can query with a sequence (run a BLAST) or with a keyword (species, 'kinase'...), so we have a lot of annotations over the sequence that correspond to text fields. The computer needs to process the keywords in the entry and compare then with your search. The searches are different, because when you are looking by sequence you need to take into account insertions, deletions, substitutions, etc. Text search can be simple, just look for an exact match or something with a regular expression. The point is, the keywords define what the sequence is doing, to which organism it belongs…

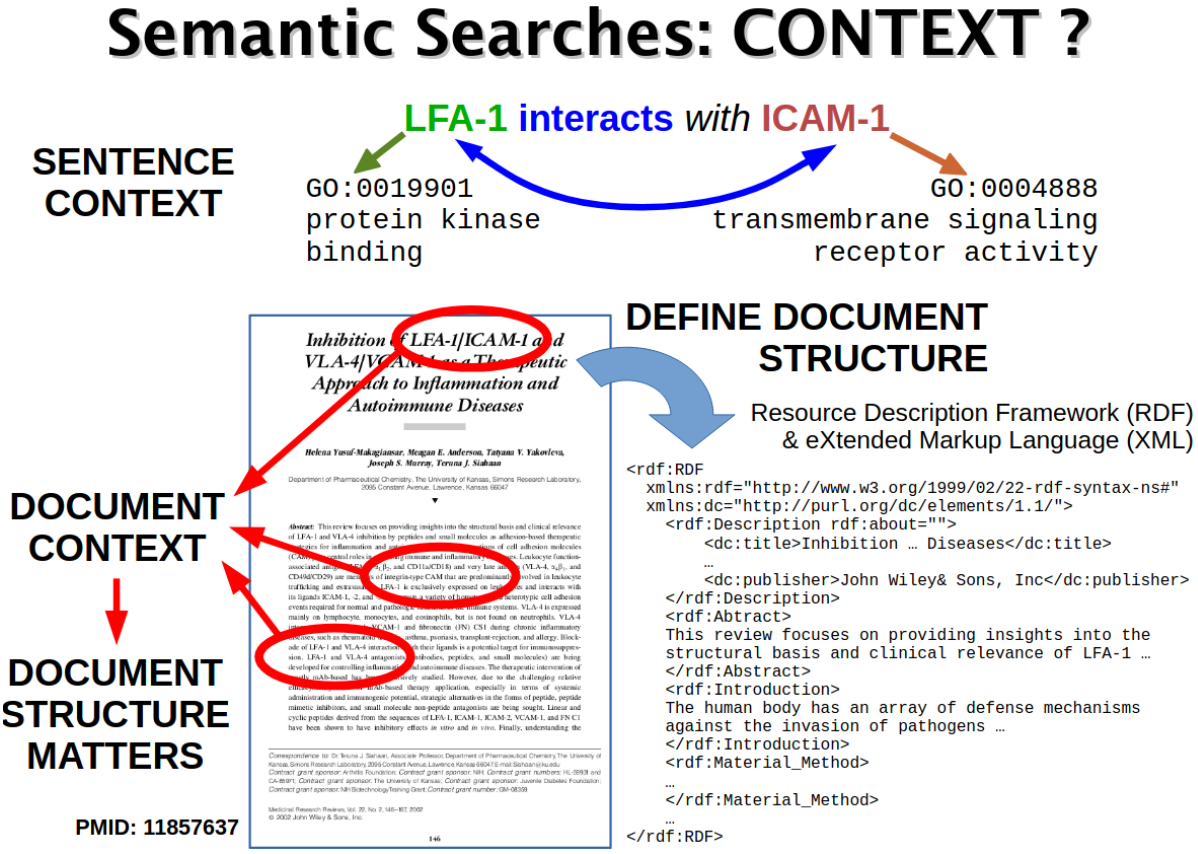
However, there are ways to improve your text search. If you search 'kinase' will only kinases appear? No, other molecules can also appear. If you only want kinases to appear, we need to define a controlled set of keywords (we need to tag kinases specifically, specific species, etc). One of the initial approaches is defining a vocabulary of terms you can use, and use them to tag the entries.

We can also do an approach to tag documents by automatically scanning all the words and seeing the frequencies that they appear. If a document mentions proteins a lot, it is likely a document about proteins. Combined with the controlled vocabulary, this is a good approach. This is for an exact word, but in biology we have problems with synonyms. Maybe one lab found a gene and named it, and then another lab found the protein that the gene encodes and put another name, and at the end we have 2 names for the same entity. If we are scanning an article, it's not just about the frequencies of the words, but also about the meaning of the words. We can do a syntactic approach and look at the structure of the sentences, not just the frequencies, and deconstructing it to figure out relevant elements of the sentence (the subjects).

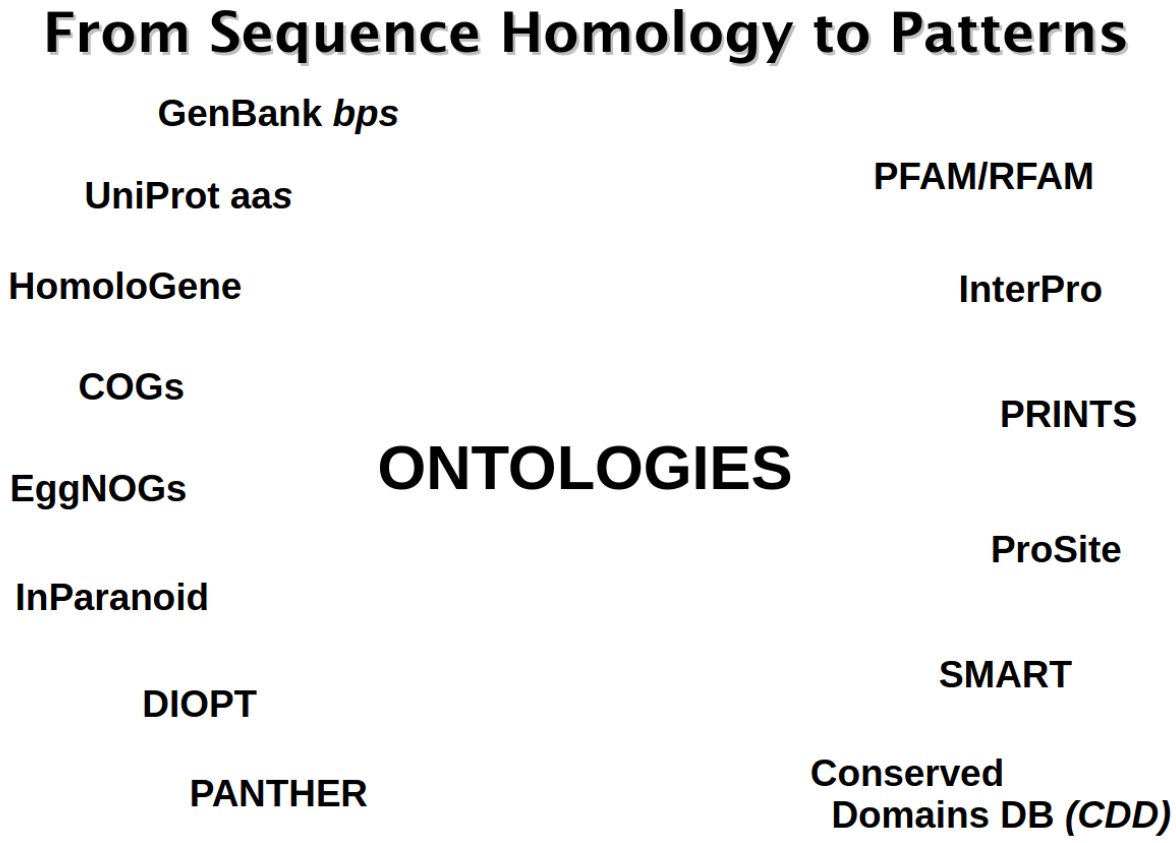


Open Biological and Biomedical Ontologies (OBO, http://obofoundry.org/)

We see those terms are synonyms but they are in the same position in the sentences, so we consider them both important. Still, this is not solving the problem of the meaning, since we still don't know if that is supposed to be a protein, a gene… First we can remove the ambiguity by using the controlled vocabulary (in this case LFA-1 will be transformed into ITGbeta2 because ITGbeta2 is the one in our vocabulary). Then we can perform a functional annotation on the word and figure out what it is exactly (ontologies). The OBO is the website where we can find all the ontologies.
Annotations are described by sequence ontology (explained in another class) and functions are described by gene ontology.
Semantic searches can take advantage of the importance of the elements in sentences, meaning that not only will it classify words by frequency but also by importance (subjects in sentences will be more important for example). But this is still not enough, because not every section of the document (title, abstract, introduction, methods, etc) has the same importance.

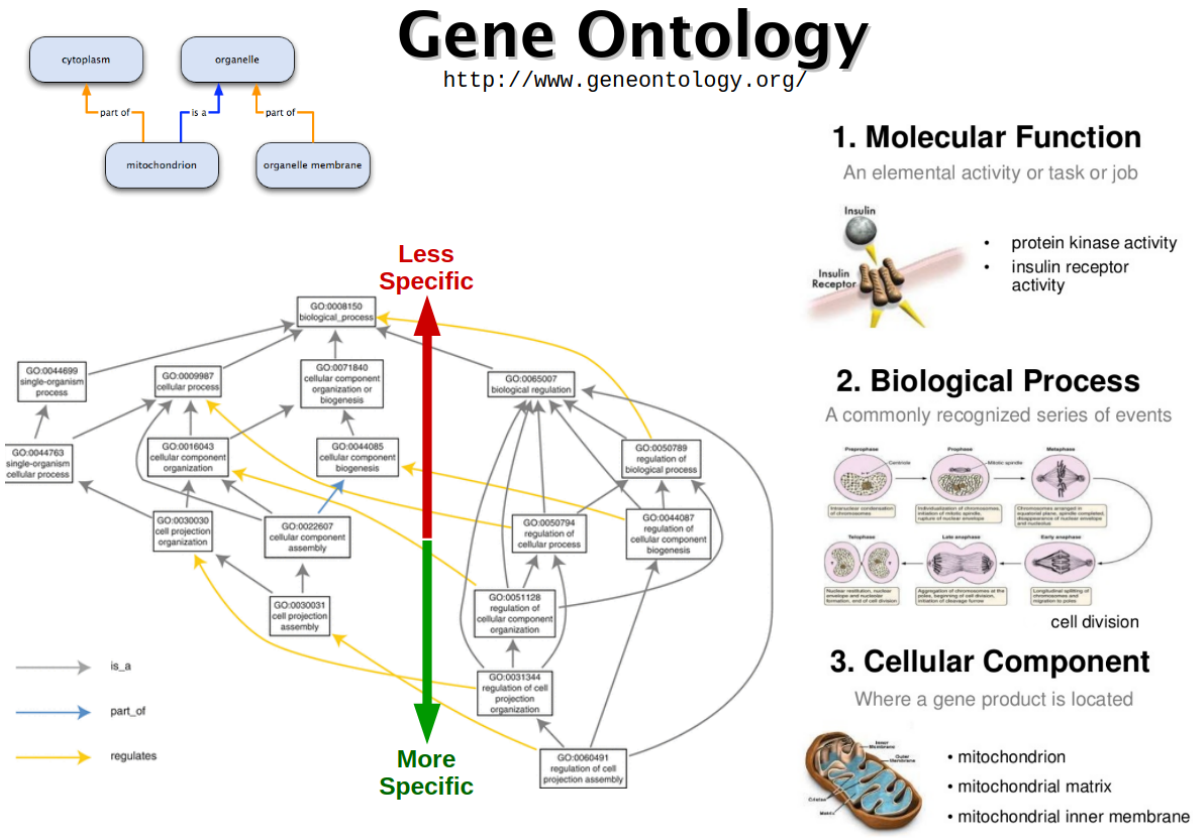


Something in the conclusions and results will be more important. And we can also get more information by looking in the PDF & XML structure of the document.
There are different ontologies depending on the database.

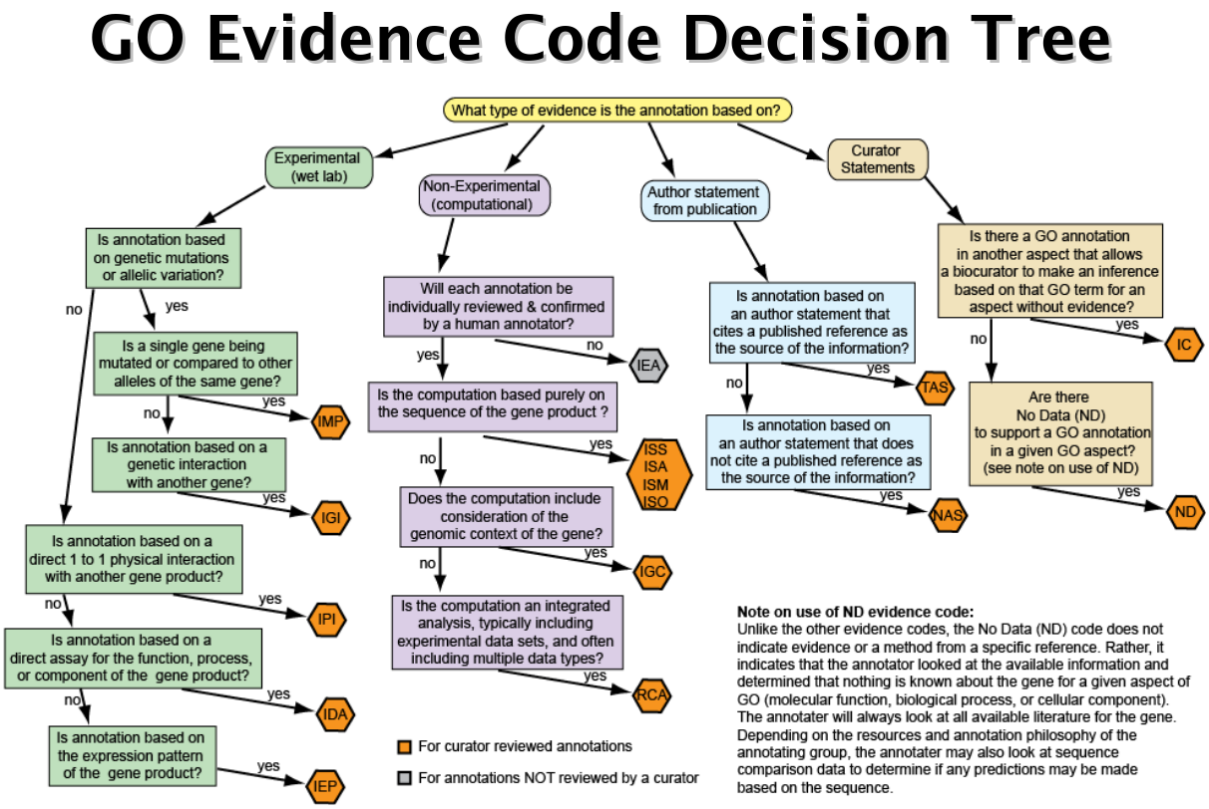


The gene ontology, in fact, is 3 ontologies at once. It is molecular function + biological processes + cellular component.

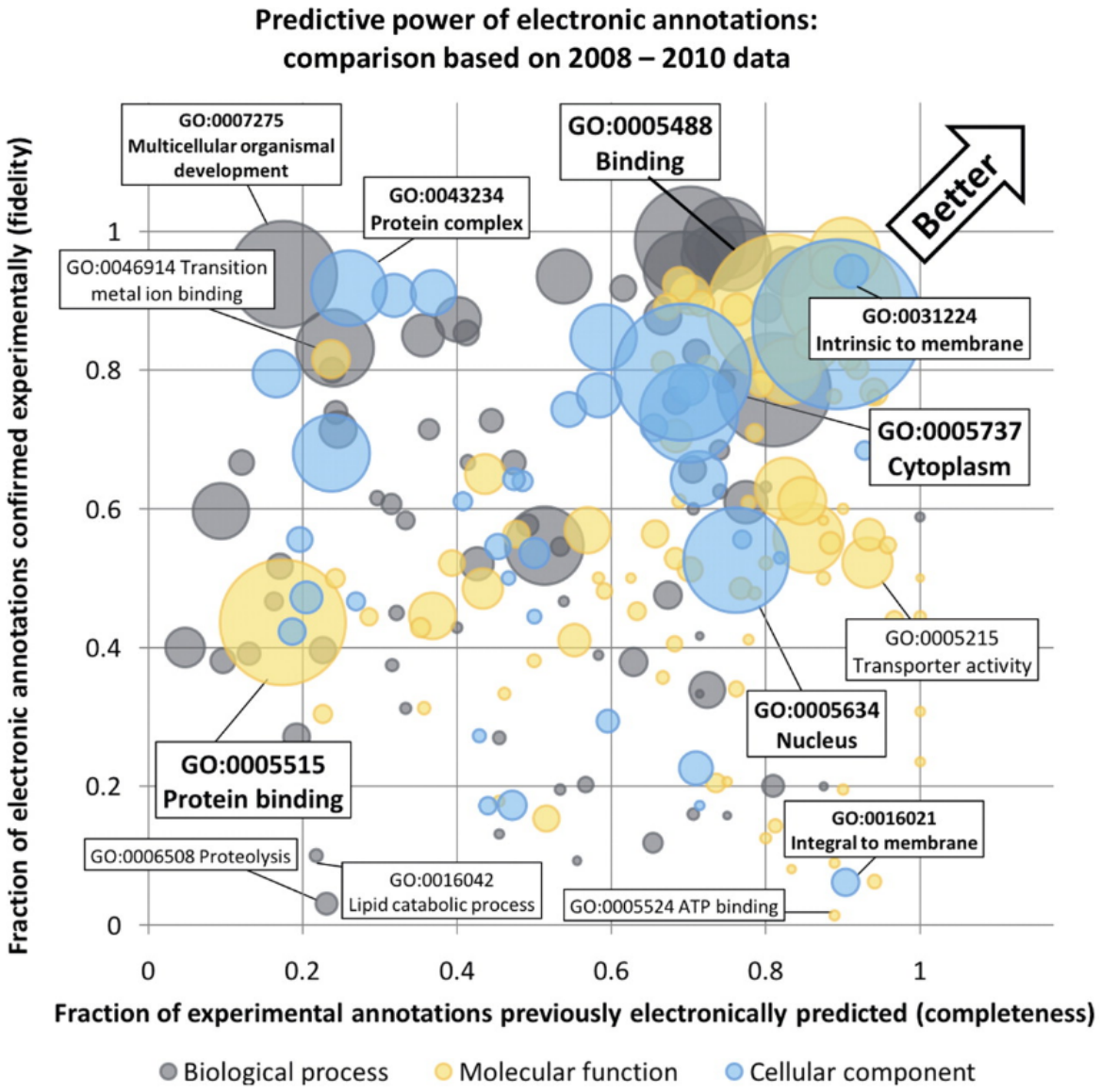
- Molecular function: Chemical activity (kinase…)
- Biological process: In which pathway it is performed (regulatory, metabolic…)
- Cellular component: Where the pathway occurs (mitochondria, nucleus…)



We also go from more specific ontologies to less specific ontologies. The more we know about a protein, the more specific we will get with the ontologies. The ideal thing is having a child at the end that perfectly defines the behavior. This is not really a tree, this is a graph, since we can have a node with multiple parents. This is an example of a decision tree of how the annotators of the gene ontology define the terms:

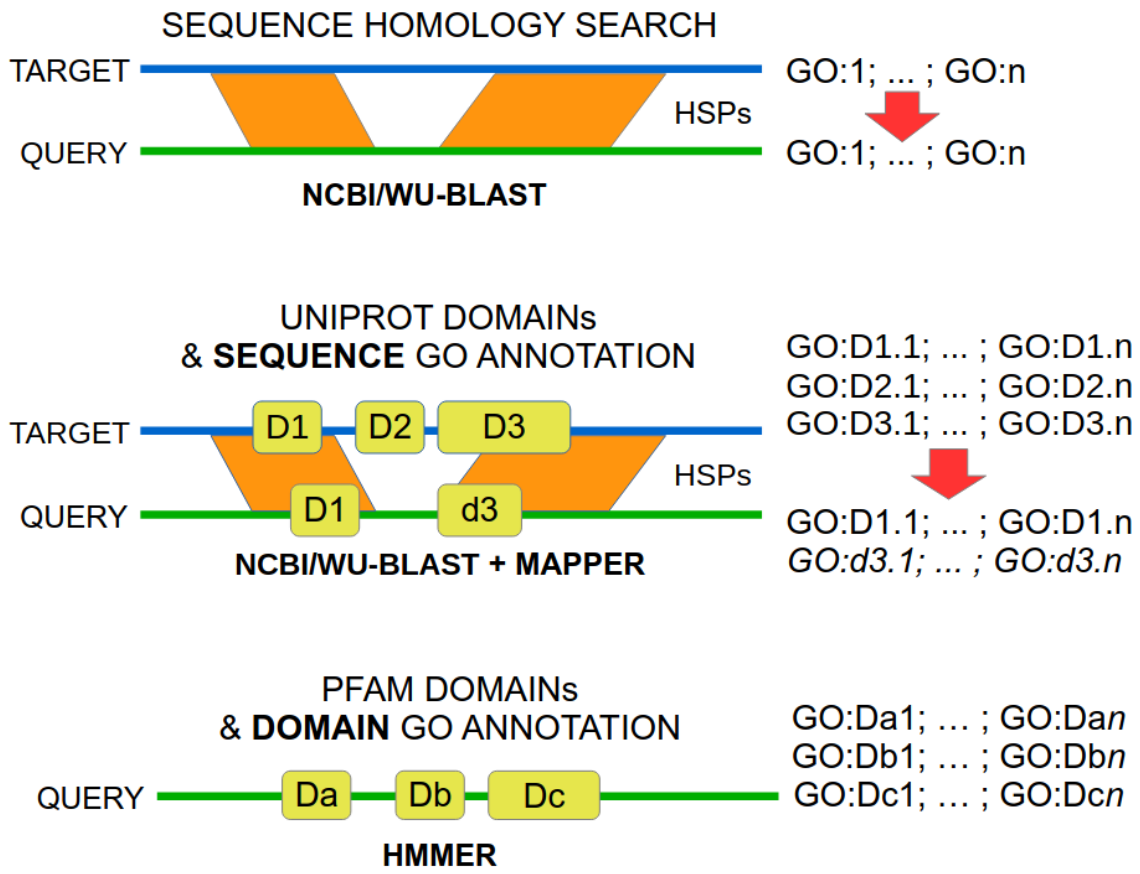


However we can have biases in the gene ontology, because it depends on the known functions. The annotation of the functions on the sequence can be biased because we don't have the same number of representatives in all the functions in the database. Here are the fractions of the elements that are validated experimentally, and the size depends on the number of genes. There are some functions that are less annotated than others and it depends how much people work with that sequence.



Once we have annotated features with a control vocabulary and a way to search effectively with IDs (numbers are faster for computers than strings), we can annotate new sequences with the known functions. We get a new sequence, and we can categorize if it is a kinase or not for example.

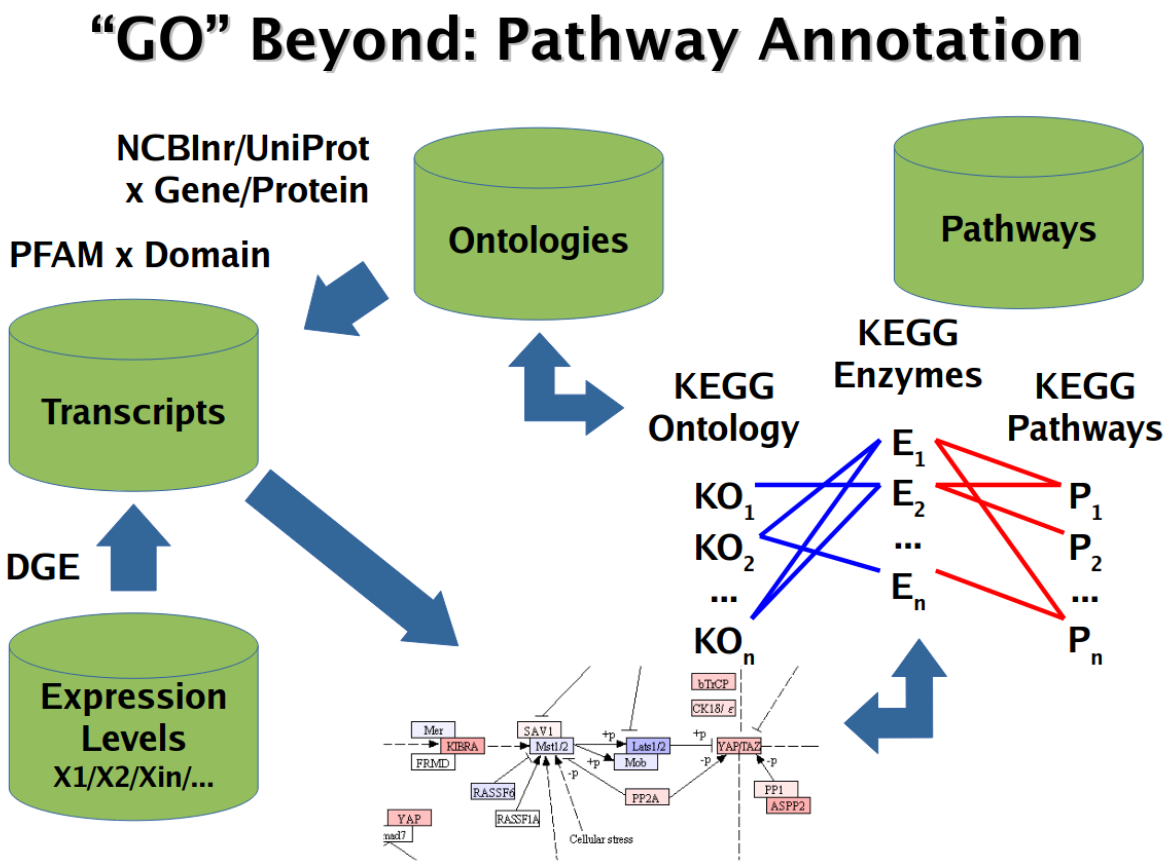
Transferring GO Annotation



We can use BLAST on the sequence and see if it is a homolog to a kinase sequence. This will be evidence at homology, so no experimental evidence yet. This is always annotated, so if we see a sequence it will say if it is “computationally predicted”, “homology-based”, “experimental evidence”...
That is a way to “transfer” the GO terms of a known sequence to an unknown sequence. But we have a problem, because the alignments can skip regions that have a domain and is not in the other protein, and maybe it can be important. Maybe a protein can be homologous but it lacks a kinase domain, which means that it cannot be a kinase even if it is homologous.
That way, we can take into account the domain annotation of the targets and the annotation of the domains in the gene ontology. We project the domains, and the unknown sequence will only acquire the GO terms of the shared domains (not all the GO terms like BLAST).
We can even use a database that already has models for the domains like PFAM and use HMMER to search for the domains. And if the sequence has matches to the domains found in the database, each domain will have its own annotation, so this is the more accurate functional annotation. Each domain will have its specific GO functional annotation. This is more computationally intensive, especially if the query is very big.
When we are transferring annotations, we have to be aware that there can be mistakes. These annotations are just hypotheses, they still have to be tested. From metagenomics

projects, there are a lot of sequences that we don't know anything about and have no similar sequences, so we cannot figure out anything about.

Pathway annotations are on top of the GO. We know that the biological process defines an annotation for a pathway, so we have databases of pathways for KEGG. So if the function is linked to a GO, we can map the GO to the pathway and get relationships between different sequences with different functions. So, we map the ontologies of the GO against the ontologies of the pathway. In KEGG, we can provide the levels of expression of the genes in the pathway by linking the GO of our own gene with the GO of the pathway, that way we are integrating our own information with the information of the pathway. So ultimately, if our gene maybe represses another one, or has some effect on a protein, we can see what is happening in our organism's pathway when our gene is overrepresented or underrepresented (we get a "full picture" of the butterfly effect). This is all done thanks to the fact that we have a controlled vocabulary.



EXPRESSION LEVEL over PATHWAYS