

## **Next Generation Sequencing Technologies: Data Analysis and Applications**

### **RNA-seq data processing pipeline**

**Dr. Riddhiman Dhar, Department of Biotechnology**

**Indian Institute of Technology, Kharagpur**

Good day, everyone. Welcome to the course on Exercise Sequencing Technologies, Data Analysis, and Applications. In the last class, we had an introduction to RNA sequencing. We talked about different methods for example, short read RNA sequencing, long read RNA sequencing, and direct RNA sequencing. So, we will now slowly get into the methods: how do you actually get the data processing right, how do you set up the pipeline, and what kind of tools and methods do we use? So, we will discuss this in theory, and alongside having some hands-on experience, I will show you the tools in this moment. And at the end, we will also have a hands-on experience with differential gene expression analysis, where we will use these tools to actually analyze a real data set and identify genes that are differentially expressed between two samples.

So, these are the concepts that we will cover in this class. So, we will talk about the RNA sequencing data processing pipeline, and in that pipeline, one of the topics that we will discuss in detail is read mapping. So, here is the pipeline for RNA sequencing data. So, we have read data in FASTQ format, we do the quality control step, or QC, as before, and then you have this mapping and assembly process, ok?

And sometimes, even after mapping, you can do an assembly to do the transcriptome assembly right. So, if you are dealing with short-read data, sometimes you would have to assemble the transcripts if you were working with these big transcripts. And after this step, we do something called quantification. So, we need to quantify how many reads are mapping to each of the genes in the genome. So, because, as we have seen, the basic principle of RNA sequencing is that the expression level of a gene is proportional to the number of reads that are mapped to that gene.

Of course, then we need to have a lot of statistical methods and models there to actually do the final quantification. Now, as we have mentioned, there are a lot of biases that are introduced when you actually do these RNA sequencing experiments. And to address these biases, we need to do

something called normalization, right? So, we will discuss these normalization methods again in the subsequent classes in much more detail. What it does is that it kind of accounts for these technical variations and biases that are introduced during sample preparation, handling, and next-generation sequencing.

So, some of these biases can be addressed computationally using different techniques. So, we will talk about these normalization methods again. There are different types of normalization methods, and each method has its own advantages and drawbacks. We will again talk about that, and we will utilize some of these normalization methods for our own analysis. And once you have done the normalization, we can then do the downstream analysis, whether we are looking at isoform discovery or whether we are going to do the gene differential gene expression analysis. So, that is what we can do after we have done the normalization.

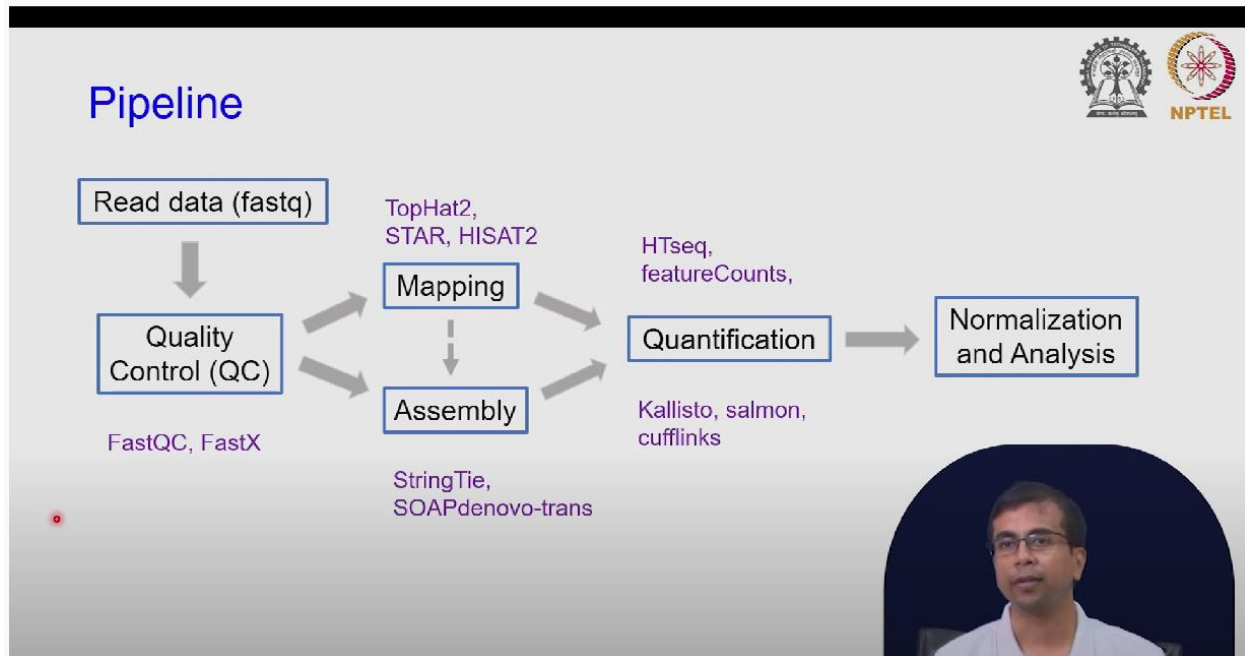
So, again, we will talk about them in the subsequent classes. So, I am just mentioning some of the tools that are available for each of these steps, some of the tools we have already learned, and, of course, other tools that are available. So, the first one is the quality control tools, right? So, we have already used FastQC, and you can use FastX. So, similarly, the quality control tool does not change because we are dealing with the same read data.

Then we have the mapping tools for RNA-seq data, right? So, and you see this, these are different right? We have tophat or tophat 2 star or high star, and HISAT2 ok. So, I will briefly mention these mapping tools. Today, we will talk about the algorithms of one of the tools in this class. Then there are assembly tools called string ties, or you can also do something called de novo transcript of assembly without looking at the genome mapping. So, in that case, it will be a de novo assembly, and there are tools for that as well.

Once we have done this right, once we have done the mapping and assembly, we can do quantification, and there are tools again that will help in this process. There are HTSeq, or feature counts, tools that actually take the mapping data and quantify how many reads map to each gene in the genome. There are other tools like Callisto, Salvar, and Cufflinks that will actually analyze this data in some different way, and they can actually give us the quantification. So, they do not

rely on the mapping data, especially Callisto and Salvar; they actually do something called pseudo-alignment. So, they give us the quantification from the pseudo-alignment.

## Lecture 33 : RNA-seq data processing pipeline



So, they are not dependent on mapping. Again, we will talk about these tools when we talk about the quantification step. And finally, we will talk about the analysis step later on in much more detail, including normalization and analysis. So, when it comes to quality control, we have FASTQC, FASTX, or other tools. So, we have used FASTQC already.

So, I am not going to go into that again, right? You already know, and given any RNA-RISing data, you should be able to do the FASTQC check yourself now, ok? So, we use the same parameters that we have checked before. Right again, we have to check such things as the read per base read quality, per sequence read quality, etcetera, adaptor contamination, etcetera. And we do data preprocessing if it is necessary, right? So, for example, if you have adaptor contamination, you want to do this trimming, or if you have some quality issues again, you want to do quality filtering, etcetera. So, as necessary, you have to decide on these data preprocessing steps.

Again, we discussed this in much more detail when we talked about genome data processing. So,

I am not going to go into that again. So, what I am going to do is I am going to discuss about read mapping ok. So, why is this a problem here, or what is new here? So, we have talked about read mapping. We have talked about the tools of BOWTIE2 right, and we have utilized their tool for read mapping in a small data set.

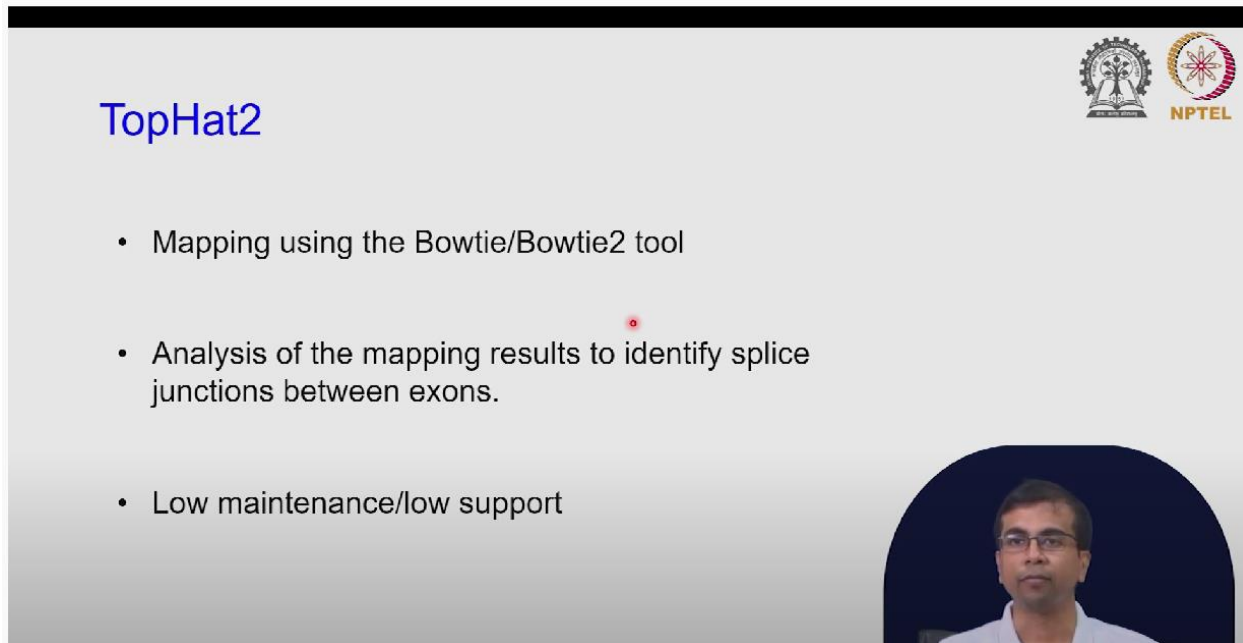
So, if you remember, BOWTIE 2 relies on Burrows-Wheeler transformation and is very efficient; it does not take too much memory. So, what is the problem? Why can't we use just BOWTIE 2 here? Now one of the major challenges with RNA-seq data is that, especially for higher eukaryotes or mammalian genomes, these transcripts contain multiple exons. So, if you look at the genes, they have multiple exons right, and in the mRNA right, what you will get is that these exons will be joined together. You can have different combinations of exons and isoforms, and then you have the reads right that come from these mRNA molecules. So, if once you take the reads right and you want to map them back to the reference genome, what will happen is that you will, in many cases, get split read alignment right.

So, what is this split read alignment or gapped alignment? It means that one part of the read will be mapping to one exon right, and another part will be mapping to another exon right. So, this is something that you need to do or need to take into account when you are mapping RNA-seq data against a reference genome. So, you have seen some examples here. If green, these are reads in green, and orange is also right; they are mapping to this exon 3 exon 4 junction. So, when you are doing this alignment or mapping, we need to do something called gapped alignment or split alignment when mapping is the reference genome. Now, you might actually be able to solve this right if you are doing this alignment against the reference transcriptome, but again, for many organisms, this reference transcriptome is not available, and also sometimes you might limit yourself right; you might not be able to discover certain isoforms of variants that are not part of the reference transcriptome.

Major challenge

Gapped alignment when mapping against the reference genome

So, in that case, you also want to align against the reference genome, and in that case, you would have to do the split read alignment; you would have to allow for that. So, that is why you need the special tools that can do this very efficiently, ok? And there are tools that have been designed to achieve this kind of alignment. So, one is called TOPHAT or TOPHAT 2. This is one of the earlier tools; it is actually based on the Bow Tie 2 program or bow tie program, and this actually utilizes kind of this extension of this bowtie using the Burrows Wheeler transformation, and it was very efficient and fast. But now that it is not used that much, they have a class of tools called HISAT or HISAT2.



The slide features the title "TopHat2" in blue text at the top left. In the top right corner, there are two logos: the Indian Institute of Technology Bombay logo and the NPTEL logo. The main content consists of three bullet points: "Mapping using the Bowtie/Bowtie2 tool", "Analysis of the mapping results to identify splice junctions between exons.", and "Low maintenance/low support". A small red dot is positioned above the second bullet point. In the bottom right corner, there is a circular video feed showing a man with glasses speaking.

- Mapping using the Bowtie/Bowtie2 tool
- Analysis of the mapping results to identify splice junctions between exons.
- Low maintenance/low support

So, this is something that is also quite widely used now, and you have STAR, of course. There could be other tools, but we are not going to discuss all of them. These are some examples and some tools that are very popular, and we can use them as well. So, TOPHAT 2 I just mentioned that this is mapping using the bow tie tool, which is only an extension of the bow tie. So, the same algorithm, the burrows wheeler transform, and all the processes are similar; if you remember the FM index and the last pass mapping, all those principles were still applied. The only thing it allowed was this gapped alignment. So, what it did was actually analyze these mapping results to identify a splice junction between exons. So, it took the same approach as the bowtie tool, but then it identified it. It looked at the mapping results very carefully, and it identified this splice junction.

So, which exons are combining with which one currently is not maintained anymore? So, this is something that you will not use, and even the authors recommend using the HISAT2 tool because HISAT2 also combines some of the features from TOPHAT. So, another tool is the STAR1, right? So, there are two steps in this process. So, it relies on something called seed searching, and then the second step is clustering, stitching and scoring.

## STAR



Two steps –

a) Seed searching

b) Clustering/stitching/scoring





So, it sounds very familiar to us: seed searching. Right, you are searching for generating seeds, and you are searching against the reference sequence. We have looked into different algorithms that will do that. And what it does is that it kind of involves something called sequential search for maximum mappable prefix, or MMP. So, how does this actually differ, and how can it incorporate this split alignment? So, here is the reason it right..


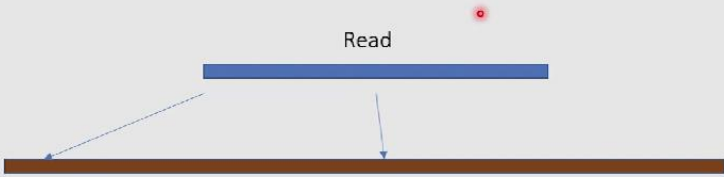
So, what this method does is start with the read right. So, it will start from this sequence in the beginning, and it will try to find some part of the sequence of the seed in the reference genome. So, this brown is the reference genome, and you see, let us say it finds the seed somewhere here, ok? So, once it finds a seed right in the reference genome, it will start extending stitching and scoring. So, we talked about this before, right?

So, what it means it is trying to extending the alignment across the genome and up to this point let us say it finds like up to this point here it finds some match and after that the match falls right the quality of alignment falls ok. So, this is what is expected for or sick data right because we need to have this capped alignment and sub-digits will not be part of the read right because they are not part of the temporary process. So, what it does is then, like, stop the alignment, and then it starts taking a seed from this region again right from here and then again starts searching in the

reference genome, ok? And let us say it finds a match here in this part of the genome and then it continues this process, and again, maybe it finds that the alignment drops after this point, right after aligning up to this point of the read, and then it will generate another seed from here, and they start searching again against the reference genome ok? So, this is where the sequential search comes in.



Seed searching involves sequential search for Maximum Mappable Prefix (MMP)

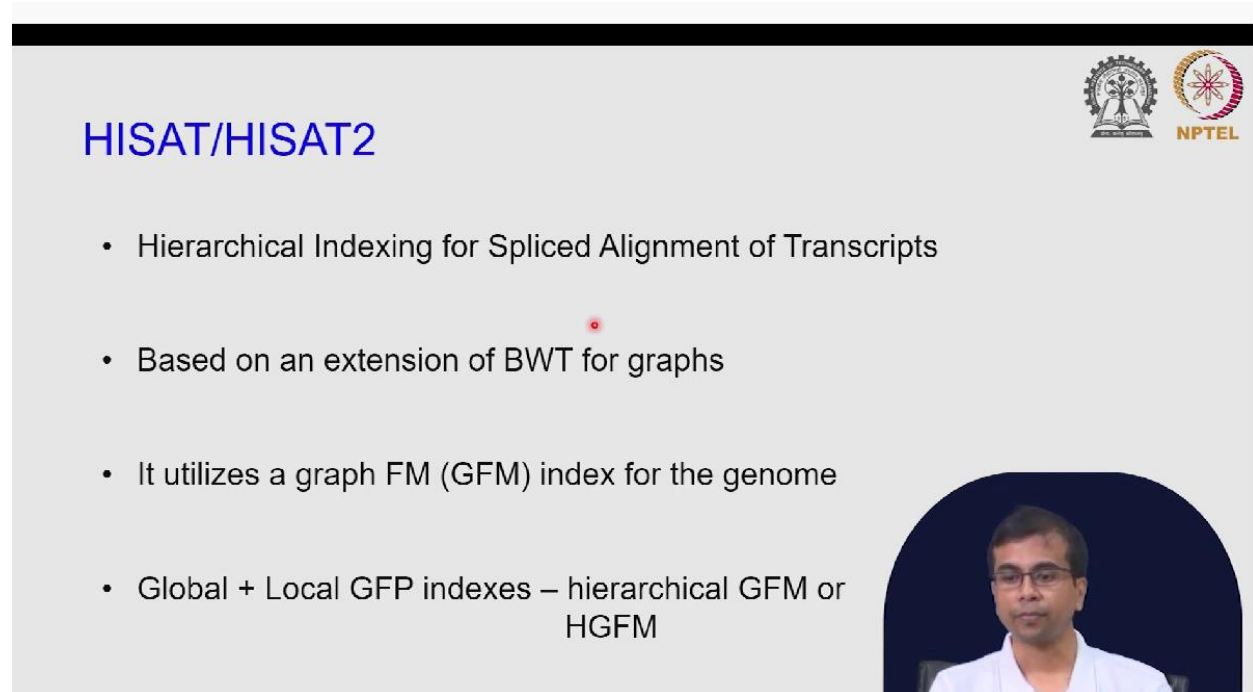


So, it is generating this prefix right in the first part of the reads, then taking the remaining part of the reads, and they are taking the first part to generate seeds and actually searching against the reference sequence. So, this tool works well with, I mean, quite a bit of efficiency, but of course, as you remember, the seed searching is quite a tedious process, and it might take a bit of computational time. So, we will talk about these two tools, HISAT and HISAT2. This is version 2 now, and it is used for read mapping in RNA sequencing data. So, the name actually comes from hierarchical indexing for spliced alignment of transcripts. As you can see, this is actually specifically designed for alignment of transcripts, and consideration of this spliced alignment is there in the tool itself. So, this is a tool that actually is based on an extension of BWT for graphs, ok?

So, we have a Burrows wheeler transform, but for graphs, ok. So, this tool relies on a graph, generates a graph, and then takes a Burrows - Wheeler transform approach for the graphs. We will



see in a moment a part of the algorithm for this tool, and it generates something very similar to this bowtie2, right where we generated this FM index. So, instead of FM index here, it generates a graph FM index right.



The slide features the title "HISAT/HISAT2" in blue text at the top left. In the top right corner, there are two logos: the Indian Institute of Technology (IIT) logo and the NPTEL logo. Below the title, there is a bulleted list of features:

- Hierarchical Indexing for Spliced Alignment of Transcripts
- Based on an extension of BWT for graphs
- It utilizes a graph FM (GFM) index for the genome
- Global + Local GFP indexes – hierarchical GFM or HGFM

In the bottom right corner, there is a circular video feed showing a man with glasses and a light blue shirt speaking.

So, it is a it is for a graph. So, that is why it is called graph FM index, or GFF, and it also does something called global indexing as well as local indexing along with some repeat indexes. So, it generates something called hierarchical GFF, or HGFF FM. So, what it can do and why it is used is that it can actually generate this graph-based data structure from the genome sequence, but on top of that, it can give SNP information and code in the graph, so it can produce a graph with the SNP information. So, what are these SNPs that are actually probably commonly found in the population? So, it can also generate this graph at 5 of these SNPs, plus it can also have the transcriptome in the graph.

## HISAT/HISAT2



Graph-based data structure from the genome sequence + SNPs + transcriptome

Two steps –

- a) Index building
- b) Read mapping



So, it can generate this complex index containing this genome sequence SNP's plus transcriptome. It can do so because it is based on a graph instead of substrings. So, it is much more flexible here again; it works in two steps. So, the first step is index building, as we have seen in Bowtie 2, right? So, it is an index building, right? It would have to do this graph structure generation, followed by the read mapping or searching, ok?

So, what we will do now is very briefly look at the algorithm, of course, without going into a lot of details about the implementation, and a part of it you will understand very easily because we have done the Bowtie2 discussion when we are discussing the genome analysis. So, how does this algorithm work? So, let us take this as an example for a reference genome: GAGCTG from the paper that actually proposed this algorithm. So, what it does is generate this graph from the reference sequence. So, here you have these six bases in the genome.

So, we will have six nodes in the graph. So, we have GAGCTG, and each node is connected by an arrow. So, if you traverse an arrow, you traverse a part of the reference sequence, okay? So, if you traverse this part, you get GAG right, and if you traverse this part, you get AGCT right. So, you traverse along the direction of the arrow, and you traverse part of the reference genome, okay?

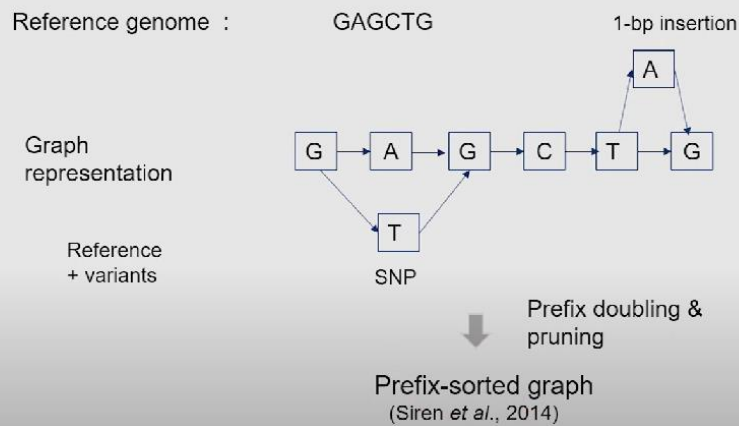
Now, what you can do is, in this graph representation of the reference genome, you can also introduce this variance.

So, here, if you imagine, you have a variant instead of this GAG. So, GAGCTG instead of that it has this GTGCTG ok. So, in that case, you have to traverse this path here, right? So, GTG, not GAG, but GTG. So, what you can do is have a node and add appropriate arrows.

So, you can traverse through this GTG path here, and you can also generate this variant genome. You can have another case here; for example, you have one base insertion, ok? So, you can do one base insertion A right. So, in that genome, So, if you are traversing that genome, you have to traverse in this direction, right CTAG, because of this insertion here.

So, what you see is that you can have this flexibility, right? You can have this deletion, you can have insertions, you can have SNPs, and you can have multi-base insertions and places as well, right? If, for example, you have this T deleted in another genome, that is a variant that is observed. So, you can simply have an arrow from C to G. So, that would be a valid path right because you can traverse then GAGC to G right, and that will give you that variant genome, ok? So, you can see these examples in this paper where they actually consider a lot of these variations, and for example, they also show how a deletion would look in this graph.

## HISAT/HISAT2 – algorithm



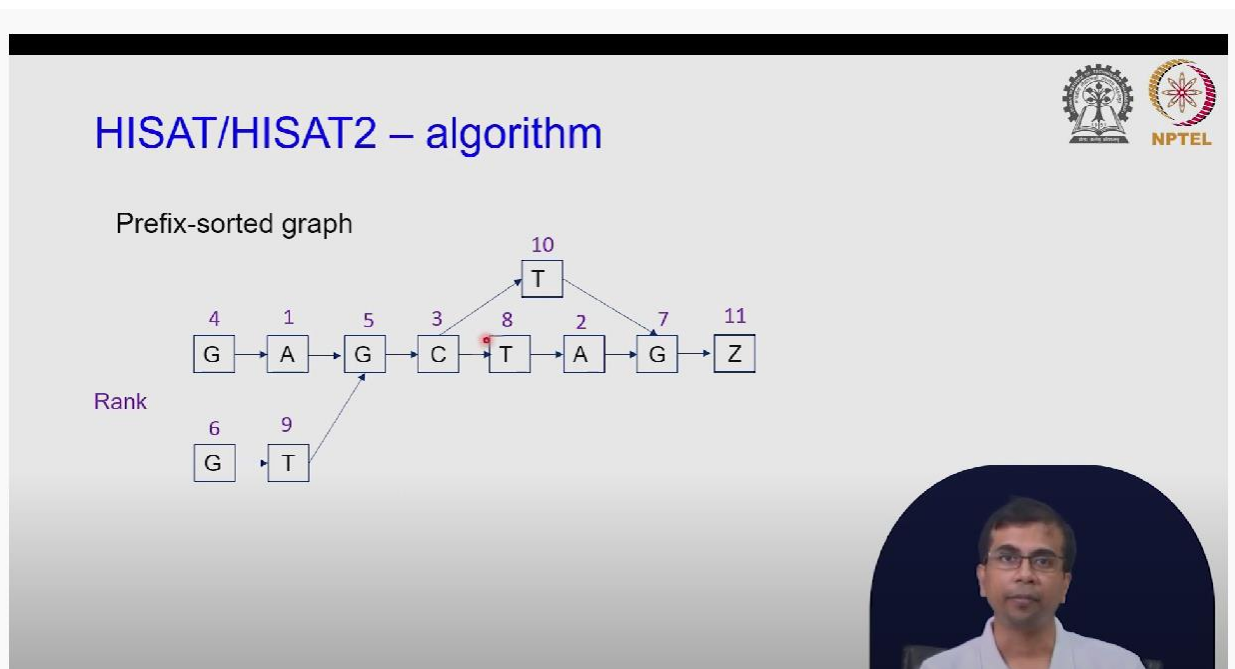
So, once you have built this graph, what you do is something called prefix doubling and pruning, and you generate something called a prefix-sorted graph. I am not going into the algorithm and how you generate this prefix sorted graph. This is discussed; this is actually developed in this paper, and they use it here to generate this prefix sorted graph, which would look something like this. Again, we have to understand the process of this prefix-sorted graph before you actually get here, ok? So, what you get is something like this, and you also have ranks added to these nodes, ok? So, what are these ranks again? So, rank 1 means any substring right that originates from this node should appear first lexicographically.

So, there is a lexicographical order again. We are coming to the Burrows-Wheeler transform right? So, there is this lexicographic sorting. So, if you have rank 1, this is the first substring that will come when you do lexicographic sorting, right? So, any substring that is originating from this node, for example, A G C T, is a substring, and this will be the lexicographically first substring if you sort all of them. And it is actually very easy to generate this rank if you just simply look at the subsequent bases and then compare against the rest of the things.

So, a is lexicographically first right. So, this will come first. So, you have two options, 1 and 2,

and among these, it is easy to decide which one is right. So, you have A G C, and here you have A G Z. So, now, notice this z was not part of the sequence. This has been added to the graph to denote that this is the end of the sequence, and this is lexicographically the highest value.

So, this will come last. So, if you have A G C and A G Z, then this one is the first one, right? So, this will get rank 1 and this will get rank 2, and then you can follow the same principle. So, you have c right; this will get rank 3; then you have g right again; you can look at the substrings that are generated from those nodes. So, you have GAGGCT right or G Z or G T right, and you can rank them accordingly, right again based on the lexical order, and then finally, you have t and then the z ok.



So, this is the last one. So, what we do next is have a tabular representation of a prefix-sorted graph, which actually talks about outgoing edges and incoming edges. And as you can see, they are actually sorted by this node rank from 1 to 11, and we have 2 columns called first and last. So, this is something that is interesting, right? So, it is very similar to what we have talked about when we discussed Paracela transform.

# HISAT/HISAT2 – algorithm



Outgoing edges			Incoming edges	
Node rank	First		Last	Node rank
1	A		G	1
2	A		T	2
3	C		G	3
	C			
4	G		Z	4
5	G		A	5
			T	
6	G		Z	6
7	G		A	7
			T	
8	T		C	8
9	T		G	9
10	T		C	10
11	Z		G	11
	Z			

• Tabular representation of Prefix-sorted graph



et al., 2019

So, what is this first and last right? So, let us consider this rank 1 right. So, this node rank 1 means this is here, right? So, this is the first node, right? So, we are simply writing the nodes in the first column. So, for the first rank, it is the outgoing edges, the first set of columns here, and the first 2 columns there under the outgoing edges.

So, we have the A here. So, the first rank is a, the second one is also a right, and the third one is C right, but since it has two outgoing edges, we have to write it twice. So, that is why we are writing the C and C OK. We just have one c in the graph, but because there are two outgoing edges, we would have to write this twice: C. Then you have 4 5 6 7; these are all G's and T's, and then finally, 11 right, which is the z here, ok? So, why do you write it twice? Because if you remember this Burrows Wheeler, transform this rotation, right?

So, what does it mean? So, j. So, this is connected to both of these G's here, ok? So, you have G here. You have G here. So, both of the G's are right. So, this j is connected to both of them. So, you have outgoing edges. You can imagine these outgoing edges; these are not actual ones, but because these are, that's right.

So, this means we are connected to the start, right? So, this is the letter that occurs. G is the first, right-first letter of these strings. So, this z is connected to these 2 G's, ok? And so, this completes the outgoing edge part. Now you have the incoming edges, and again, we have now given this node ranks 1-2.

What are we doing here? So, we are actually writing the corresponding last or the corresponding incoming edge for this first node, ok? So, for this node, what is the incoming node? So, this incoming node actually comes from G here, right? So, that is why this base is G-ok. So, the last node that was visited right before this would be this.

So, G would be the node that comes before that. Similarly, for this one, a right, what is the last node, which is T right? You can see this right. So, before A, you have T. So, that is why you have written T here, okay? And we can now continue right for this C node. Even though we have written it down twice, it is always G right that comes before this C. And for the incoming edge, we just write it once because we have just one incoming edge.

In some cases, for example, this 5 here right this G 5 here right, you have two incoming nodes; you have one from the left and another from the right. So, that is why you have written it down twice: A T for G 6, right? So, this one here, you see, we have written it down as z because, as we have talked about, there is a connection from Z to G, which we imagine is okay. And we can complete this table this way, right? So, we have completed this right, and what you get is this last first mapping property ok?

## HISAT/HISAT2 – algorithm



Outgoing edges			Incoming edges	
Node rank	First		Last	Node rank
1	A		G	1
2	A		T	2
3	C		G	3
	C			
4	G		Z	4
5	G		A	5
	G		T	
6	G		Z	6
7	G		A	7
			T	
8	T		C	8
9	T		G	9
10	T		C	10
11	Z		G	11
	Z			

Last-First Mapping!



et al., 2019

And you remember the first mapping, right? So, the first occurrence of a letter in the first column and the last occurrence of that letter in the last column correspond to the same letter in the reference tree, ok? So, we talked about this in the burrow-wheeler transformation. So, this is what applies here, right? So, this G corresponds to this G because of this last mapping property, ok? So, what you can do now is utilize this last part mapping property in this table, and we can map certain reads.



## HISAT/HISAT2 – algorithm



Tabular representation of Prefix-sorted graph

- Global + Local GFP indexes – hierarchical GFM or HGFM



And what you see here that we have seen is that it is very flexible, right? You can actually accommodate mutations. You can accommodate this splice variance right because of these graphs, right because you have converted this reference into a graph. So, we have these global plus local indexes plus some repeat indexes, which give us this h g f m ok. So, how do you search for a substring now? So, let us take this example: we have G C T G, right?

# HISAT/HISAT2 – algorithm

Outgoing edges		Incoming edges	
Node rank	First	Last	Node rank
1	A	G	1
2	A	T	2
3	C C	G	3
4	G	Z	4
5	G	A T	5
6	G	Z	6
7	G	A T	7
8	T	C	8
9	T	G	9
10	T	C	10
11	Z Z	G	11

Searching a substring

GCTG



et al., 2019

# HISAT/HISAT2 – algorithm

Outgoing edges		Incoming edges	
Node rank	First	Last	Node rank
1	A	G	1
2	A	T	2
3	C C	G	3
4	G	Z	4
5	G	A T	5
6	G	Z	6
7	G	A T	7
8	T	C	8
9	T	G	9
10	T	C	10
11	Z Z	G	11

Searching a substring

GCTG



et al., 2019

# HISAT/HISAT2 – algorithm

Outgoing edges		Incoming edges		
Node rank	First		Last	Node rank
1	A		G	1
2	A		T	2
3	C C		G	3
4	G		Z	4
5	G		A T	5
6	G		Z	6
7	G		A T	7
8	T		C	8
9	T		G	9
10	T		C	10
11	Z Z		G	11

Searching a substring

GCTG



et al., 2019

# HISAT/HISAT2 – algorithm



Outgoing edges			Incoming edges	
Node rank	First		Last	Node rank
1	A		G	1
2	A		T	2
3	C		G	3
	C			
4	G		Z	4
5	G		A	5
			T	
6	G		Z	6
7	G		A	7
			T	
8	T		C	8
9	T		G	9
10	T		C	10
11	Z		G	11
	Z			

Searching a substring

GCTG



et al., 2019

# HISAT/HISAT2 – algorithm



Outgoing edges			Incoming edges	
Node rank	First		Last	Node rank
1	A		G	1
2	A		T	2
3	C		G	3
	C			
4	G		Z	4
5	G		A	5
			T	
6	G		Z	6
7	G		A	7
			T	
8	T		C	8
9	T		G	9
10	T		C	10
11	Z		G	11
	Z			

Searching a substring

GCTG



et al., 2019

So, we can start with the last one, the G, and see where these G's are. So, these G's are here; their correspondence is given by these red arrows, and we can go to the last column right. So, which one is the letter that occurs before g? Remember, this is the principle that we had in the first mapping. So, the T is the letter that occurs before G, and we have found T here in two cases. So, we need to follow up on those two parts and this T because, because of the last mapping, they

correspond to these two T's here in the first column right.

And only for one of them can we find C, right? We are looking for C. So, we can find C, and this C corresponds to the one there right? This is the second C, and then we have G, ok? So, we have found the substring G C T G ok. So, it follows the same way that we have discussed before, and we can find any substring that we want. So, what you can now imagine is that we can extend this graph to generate a graph with splice variance, and we can do split field alignment because of the flexibility that is built into this algorithm.

graph-based alignment of next generation sequencing reads to a population or genomes

## Manual

### Introduction

#### What is HISAT2?

HISAT2 is a fast and sensitive alignment program for mapping next-generation sequencing reads (whole-genome, transcriptome, and exome sequencing data) against the general human population (as well as against a single reference genome). Based on [GCSA](#) (an extension of [BWT](#) for a graph), we designed and implemented a graph FM index (GFM), an original approach and its first implementation to the best of our knowledge. In addition to using one global GFM index that represents general population, HISAT2 uses a large set of small GFM indexes that collectively cover the whole genome (each index representing a genomic region of 56 Kbp, with 55,000 indexes needed to cover human population). These small indexes (called local indexes) combined with several alignment strategies enable effective alignment of sequencing reads. This new indexing scheme is called Hierarchical Graph FM index (HGFM). We have developed HISAT 2 based on the [HISAT](#) and [Bowtie2](#).

**Funding**

This work was supported in part by the National Genome Research Institute under grants R01-L10006677, and NIH grants R01-L1006845, GM083873 and NSF grant CCF-0347892 to Steve

genome	<a href="https://genome-idx.s3.amazonaws.com/hisat/bdgp6.tar.gz">https://genome-idx.s3.amazonaws.com/hisat/bdgp6.tar.gz</a>
genome_tran	<a href="https://genome-idx.s3.amazonaws.com/hisat/bdgp6_tran.tar.gz">https://genome-idx.s3.amazonaws.com/hisat/bdgp6_tran.tar.gz</a>
• UCSC dm6	
genome	<a href="https://genome-idx.s3.amazonaws.com/hisat/dm6.tar.gz">https://genome-idx.s3.amazonaws.com/hisat/dm6.tar.gz</a>
<b>C. elegans</b>	
• WBcel235	
genome	<a href="https://genome-idx.s3.amazonaws.com/hisat/wbcel235.tar.gz">https://genome-idx.s3.amazonaws.com/hisat/wbcel235.tar.gz</a>
genome_tran	<a href="https://genome-idx.s3.amazonaws.com/hisat/wbcel235_tran.tar.gz">https://genome-idx.s3.amazonaws.com/hisat/wbcel235_tran.tar.gz</a>
• UCSC ce10	
genome	<a href="https://cloud.biohpc.swmed.edu/index.php/s/bbynxoY2TPpRNQb/download">https://cloud.biohpc.swmed.edu/index.php/s/bbynxoY2TPpRNQb/download</a>
<b>S. cerevisiae</b>	
• R64-1-1	
genome	<a href="https://cloud.biohpc.swmed.edu/index.php/s/JRSokHD5cHfpCFE/download">https://cloud.biohpc.swmed.edu/index.php/s/JRSokHD5cHfpCFE/download</a>
genome_tran	<a href="https://cloud.biohpc.swmed.edu/index.php/s/akeIMrGGtt5KoJY/download">https://cloud.biohpc.swmed.edu/index.php/s/akeIMrGGtt5KoJY/download</a>
• UCSC sacCer3	
genome	<a href="https://cloud.biohpc.swmed.edu/index.php/s/Gsq4goLW4TDz4E/download">https://cloud.biohpc.swmed.edu/index.php/s/Gsq4goLW4TDz4E/download</a>

So, these are the references that we have used. So, what I wanted to do now is just show you the HiSAT2 program right before we actually conclude this class. So, I will go to this HiSAT 2 program now, ok? So, let us see right ok. So, here is the website where you can actually get this program from, and you have the manual you have download options on the right-hand side; you can see the download option; you can download any version of this one; and also, as you will see, you have some of the index files that are already built and available.

## Binaries

### Version: HISAT2 2.2.1

**Release Date:** 7/24/2020

Source	<a href="https://cloud.biohpc.swmed.edu/index.php/s/fe9QC5X3NH4QwBi/download">https://cloud.biohpc.swmed.edu/index.php/s/fe9QC5X3NH4QwBi/download</a>
OSX_x86_64	<a href="https://cloud.biohpc.swmed.edu/index.php/s/zMgEtnF6LjnJFrr/download">https://cloud.biohpc.swmed.edu/index.php/s/zMgEtnF6LjnJFrr/download</a>
Linux_x86_64	<a href="https://cloud.biohpc.swmed.edu/index.php/s/oTtGWbWjx5Q2Ho/download">https://cloud.biohpc.swmed.edu/index.php/s/oTtGWbWjx5Q2Ho/download</a>

### Version: HISAT2 2.2.0

**Release Date:** 2/6/2020

Source	<a href="https://cloud.biohpc.swmed.edu/index.php/s/hisat2-220-source/download">https://cloud.biohpc.swmed.edu/index.php/s/hisat2-220-source/download</a>
OSX_x86_64	<a href="https://cloud.biohpc.swmed.edu/index.php/s/hisat2-220-OSX_x86_64/download">https://cloud.biohpc.swmed.edu/index.php/s/hisat2-220-OSX_x86_64/download</a>
Linux_x86_64	<a href="https://cloud.biohpc.swmed.edu/index.php/s/hisat2-220-Linux_x86_64/download">https://cloud.biohpc.swmed.edu/index.php/s/hisat2-220-Linux_x86_64/download</a>

### Version: HISAT2 2.1.0

So, here is the here are the versions of for you have the for OSX or for Linux right. So, you can download any version that is appropriate for your system's operating system. So, in this system, we are running WSL. So, we can use the Linux version, and we can use any of these binaries in any of these versions of HiSAT 2. In addition, you will see some index files that have already been built.

So, this is very useful, right? So, which means you can skip the first step if you are working on any of these organisms and you have this index file available, you can skip the first step when you are running this program, right? So, you have this already-given genome and transcriptome, right? So, you have this here genome SNP transcriptome, right? So, where do you have the SNP data and part of the index? So, if you have these files available, you can simply download them, use these index files, and do the mapping very easily.

And you have the manuals here; this kind of describes how this method is developed right and also what the running mode is, etcetera alignment summary that you will get, etcetera. So, what I will do is actually run it from the command line. So, I have actually downloaded this program. So, here, we can go to this directory, cd RNAseq\_data\_analysis, and inside this, we can see HiSAT2. So,



you will probably notice this here; it is already highlighted, right? This is the program that I have downloaded and extracted.

```
rdhar@LAPTOP-3K4C9VBI: /mnt/c/Users/Dhar/Desktop/NGS_Data_Analysis_HandsOn1/RNAseq_data_analysis$ ls
SRR6108614.fastq  hisat2-2.2.1-Linux_x86_64.zip  r64.tar.gz  r64_tran.tar.gz
rdhar@LAPTOP-3K4C9VBI: /mnt/c/Users/Dhar/Desktop/NGS_Data_Analysis_HandsOn1/RNAseq_data_analysis$ |
```

```
rdhar@LAPTOP-3K4C9VBI: /mnt/c/Users/Dhar/Desktop/NGS_Data_Analysis_HandsOn1/RNAseq_data_analysis$ ls
SRR6108614.fastq  hisat2-2.2.1-Linux_x86_64.zip  r64.tar.gz  r64_tran.tar.gz
rdhar@LAPTOP-3K4C9VBI: /mnt/c/Users/Dhar/Desktop/NGS_Data_Analysis_HandsOn1/RNAseq_data_analysis$ ./hisat2-2.2.1-Linux_x86_64/hisat2-2.1/
AUTHORS          hisat2-align-l-debug          hisat2-inspect-s-debug
LICENSE          hisat2-align-s                hisat2-repeat
MANUAL           hisat2-align-s-debug         hisat2-repeat-debug
MANUAL.markdown hisat2-build                  hisat2_extract_exons.py
NEWS             hisat2-build-l              hisat2_extract_snps_haplotypes_UCSC.py
TUTORIAL        hisat2-build-l-debug        hisat2_extract_snps_haplotypes_VCF.py
VERSION         hisat2-build-s              hisat2_extract_splice_sites.py
example/        hisat2-build-s-debug        hisat2_read_statistics.py
extract_exons.py hisat2-inspect               hisat2_simulate_reads.py
extract_splice_sites.py hisat2-inspect-l            scripts/
hisat2          hisat2-inspect-l-debug
hisat2-align-l hisat2-inspect-s
rdhar@LAPTOP-3K4C9VBI: /mnt/c/Users/Dhar/Desktop/NGS_Data_Analysis_HandsOn1/RNAseq_data_analysis$ ./hisat2-2.2.1-Linux_x86_64/hisat2-2.1/hisat2|
```

And I can simply go inside after extraction. I can go and call this program OK. One of the things you might want to check is whether you have permission to run otherwise, you can change permission through `chmod` ok. So, this is very simple; actually, once you download the program, you can simply access it, and you can also say `minus minus help`. You can simply call this from here, or you can add it to `Path`. Right in the last hands-on, we discussed how we can add this program to `Path`, and you can do that as well. So, you can simply call `HiSAT2`, and it will be available everywhere in the system, ok? So, I will just simply run `high set 2 minus minus help` and you will see certain options here, and what you will notice right away is that the usage is very similar to the program that we use, `bowtie2`.



```
rdhar@LAPTOP-3K4C9VBI:/mnt/c/Users/Dhar/Desktop/NGS_Data_Analysis_HandsOn1/RNAseq_data_analysis$ ./hisat2-2.2.1-Linux_x86_64/hisat2-2.1/hisat2 --help
HISAT2 version 2.2.1 by Daehwan Kim (infphilo@gmail.com, www.ccb.jhu.edu/people/infphilo)
Usage:
  hisat2 [options]* -x <ht2-idx> {-1 <m1> -2 <m2> | -U <r> | --sra-acc <SRA accession number>} [-S <sam>]

<ht2-idx> Index filename prefix (minus trailing .X.ht2).
<m1>       Files with #1 mates, paired with files in <m2>.
           Could be gzip'ed (extension: .gz) or bzip2'ed (extension: .bz2).
<m2>       Files with #2 mates, paired with files in <m1>.
           Could be gzip'ed (extension: .gz) or bzip2'ed (extension: .bz2).
<r>        Files with unpaired reads.
           Could be gzip'ed (extension: .gz) or bzip2'ed (extension: .bz2).
<SRA accession number> Comma-separated list of SRA accession numbers, e.g. --sra-acc SRR353653,SRR353654.
<sam>      File for SAM output (default: stdout)

<m1>, <m2>, <r> can be comma-separated lists (no whitespace) and can be
specified many times. E.g. '-U file1.fq,file2.fq -U file3.fq'.

Options (defaults in parentheses):

Input:
-q          query input files are FASTQ .fq/.fastq (default)
--qseq     query input files are in Illumina's qseq format
-f         query input files are (multi-)FASTA .fa/.mfa
-r         query input files are raw one-sequence-per-line
-c         <m1>, <m2>, <r> are sequences themselves, not files
-s/--skip <int> skip the first <int> reads/pairs in the input (none)
-u/--upto <int> stop after first <int> reads/pairs (no limit)
-5/--trim5 <int> trim <int> bases from 5'/left end of reads (0)
-3/--trim3 <int> trim <int> bases from 3'/right end of reads (0)
--phred33  qualities are Phred+33 (default)
--phred64  qualities are Phred+64
--int-quals qualities encoded as space-delimited integers
--sra-acc  SRA accession ID
```



So, it is very similar to what we used earlier, ok? So, high set 2 minus X ht2 index. So, if this is the base name of the index 5, then you have the input file options right minus 1 m1 minus 2 m2, or unpaired data, or if you want to give an SRS accession number, this is from a database, and I can directly run that accession number and the output in SAM format. So, this is very good, because we already learned all these things with Bowtie 2. So, we can simply follow that and do the same thing here, ok? So, there are a few extra things you will probably notice.

```

Input:
-q          query input files are FASTQ .fq/.fastq (default)
--qseq     query input files are in Illumina's qseq format
-f          query input files are (multi-)FASTA .fa/.mfa
-r          query input files are raw one-sequence-per-line
-c          <m1>, <m2>, <r> are sequences themselves, not files
-s/--skip <int> skip the first <int> reads/pairs in the input (none)
-u/--upto <int> stop after first <int> reads/pairs (no limit)
-5/--trim5 <int> trim <int> bases from 5'/left end of reads (0)
-3/--trim3 <int> trim <int> bases from 3'/right end of reads (0)
--phred33  qualities are Phred+33 (default)
--phred64  qualities are Phred+64
--int-quals qualities encoded as space-delimited integers
--sra-acc  SRA accession ID

Presets:
Same as:
--fast          --no-repeat-index
--sensitive     --bowtie2-dp 1 -k 30 --score-min L,0,-0.5
--very-sensitive --bowtie2-dp 2 -k 50 --score-min L,0,-1

Alignment:
--bowtie2-dp <int> use Bowtie2's dynamic programming alignment algorithm (0) - 0: no dynamic programming, 1: conditional dynamic programming, and 2: unconditional dynamic programming (slowest)
--n-ceil <func>   func for max # non-A/C/G/Ts permitted in aln (L,0,0.15)
--ignore-quals   treat all quality values as 30 on Phred scale (off)
--nofw           do not align forward (original) version of read (off)
--norc          do not align reverse-complement version of read (off)
--no-repeat-index do not use repeat index

Spliced Alignment:
--pen-cansplice <int> penalty for a canonical splice site (0)
--pen-noncansplice <int> penalty for a non-canonical splice site (12)
--pen-canintronlen <func> penalty for long introns (G,-8,1) with canonical splice sites
--pen-noncanintronlen <func> penalty for long introns (G,-8,1) with noncanonical splice sites
--min-intronlen <int> minimum intron length (20)

```



```

Spliced Alignment:
--pen-cansplice <int> penalty for a canonical splice site (0)
--pen-noncansplice <int> penalty for a non-canonical splice site (12)
--pen-canintronlen <func> penalty for long introns (G,-8,1) with canonical splice sites
--pen-noncanintronlen <func> penalty for long introns (G,-8,1) with noncanonical splice sites
--min-intronlen <int> minimum intron length (20)
--max-intronlen <int> maximum intron length (500000)
--known-splicesite-infile <path> provide a list of known splice sites
--novel-splicesite-outfile <path> report a list of splice sites
--novel-splicesite-infile <path> provide a list of novel splice sites
--no-temp-splicesite          disable the use of splice sites found
--no-spliced-alignment       disable spliced alignment
--rna-strandness <string> specify strand-specific information (unstranded)
--tmo                        reports only those alignments within known transcriptome
--dta                        reports alignments tailored for transcript assemblers
--dta-cufflinks              reports alignments tailored specifically for cufflinks
--avoid-pseudogene           tries to avoid aligning reads to pseudogenes (experimental option)
--no-templaten-adjustment    disables template length adjustment for RNA-seq reads

```

```

Output:
-t/--time      print wall-clock time taken by search phases
--un <path>   write unpaired reads that didn't align to <path>
--al <path>   write unpaired reads that aligned at least once to <path>
--un-conc <path> write pairs that didn't align concordantly to <path>
--al-conc <path> write pairs that aligned concordantly at least once to <path>
(Note: for --un, --al, --un-conc, or --al-conc, add '-gz' to the option name, e.g.
--un-gz <path>, to gzip compress output, or add '-bz2' to bzip2 compress output.)
--summary-file <path> print alignment summary to this file.
--new-summary    print alignment summary in a new style, which is more machine-friendly.
--quiet          print nothing to stderr except serious errors
--met-file <path> send metrics to file at <path> (off)
--met-stderr     send metrics to stderr (off)
--met <int>     report internal counters & metrics every <int> secs (1)
--no-head        suppress header lines, i.e. lines starting with @
--no-sq          suppress @SQ header lines
--rg-id <text>  set read group id, reflected in @RG line and RG:Z: opt field
--rg <text>     add <text> ("lab:value") to @RG line of SAM header.
Note: @RG line only printed when --rg-id is set.
--omit-sec-seq  put '*' in SEQ and QUAL fields for secondary alignments.

```

So, when you come down to these options, right? So, the options again look very similar to what

we have discussed before, right? So, you have this thread 33, etcetera, with all these options: trim 5, trim 3, etcetera, fast sensitive, very sensitive, these options again, right? So, these kinds of things are there, but along with that, what you will see now is another set of options for spliced alignment. So, you can see this right whether you want a novel spliced site out file, or if you say no spliced alignment, you do not want this spliced alignment because you are mapping to a reference transcriptome.

So, you do not need this. So, you can also have this option. So, these options are actually extra compared to Bowtie2, and then finally, the output is also in an SAM file, which means we know the SAM format and can interpret the data very easily. So, we will use these tools right later on. When we do the hands-on, I will demonstrate further, and we can go back to the conclusion, ok? So, to conclude, right. So, we have discussed the read mapping step and some of the algorithms that are used for this process, and this is one of the first critical steps of the RNA sequencing data processing pipeline. The challenge we have discussed is that, in many cases, we need this split read mapping right if you are aligning against the reference genome.

So, we cannot just use Bowtie2; we need some sort of other modification to that algorithm. We talked about this STAR and HiSAT 2 bit in detail, and we have seen that HiSAT 2 uses a BWT for graphs. So, the Burrows Wheeler transform for graphs creates a very similar HGFM index for read mapping, ok? And it utilizes this HGFM index to actually search for splice variants and reads in the reference genome data. Thank you.