Next Generation Sequencing Technologies: Data Analysis and Applications

Read Mapping

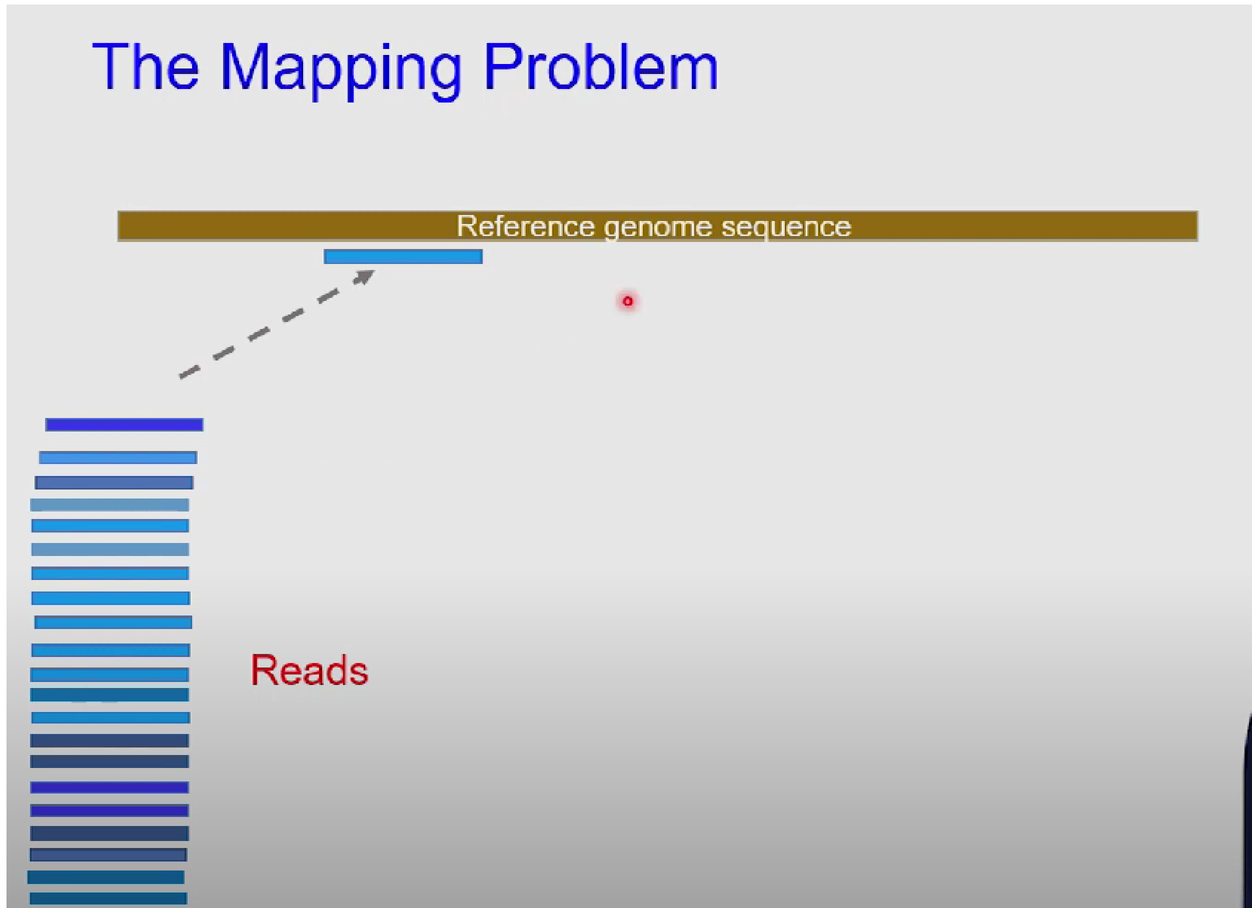
Dr. Riddhiman Dhar, Department of Biotechnology

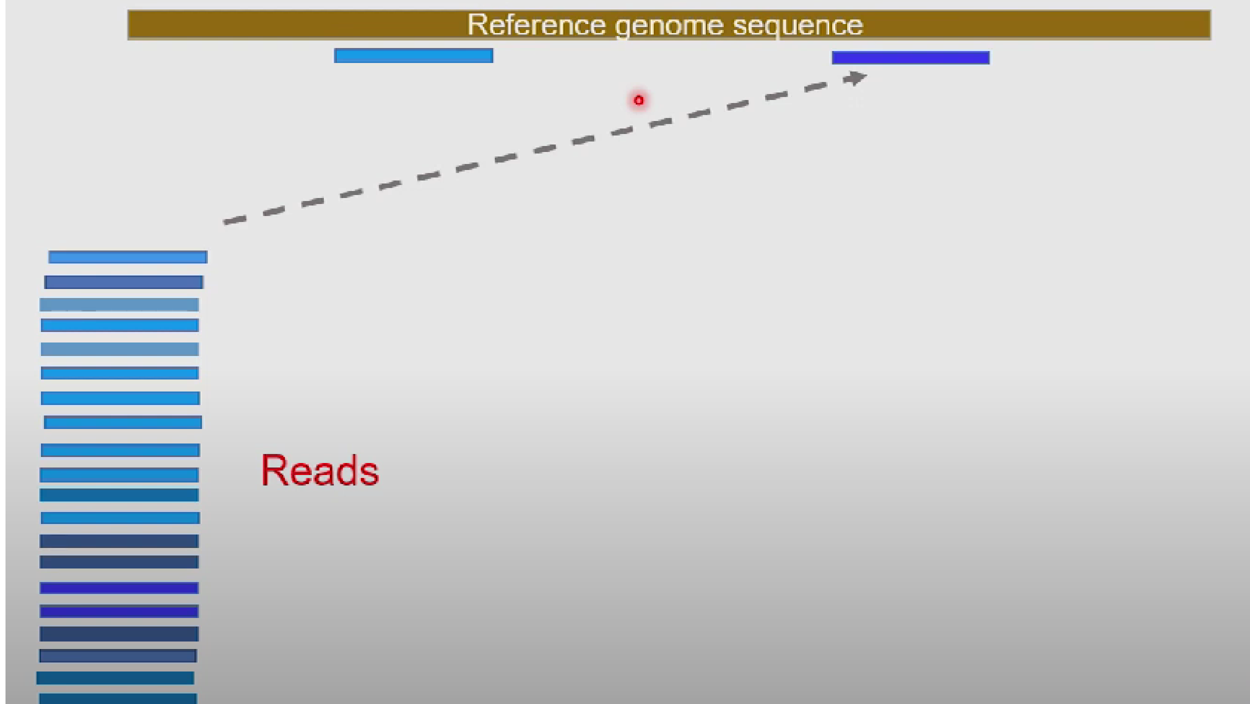
Indian Institute of Technology, Kharagpur

Good day, everyone. Welcome to the course on Next Generation Sequencing Technologies, Data Analysis, and Applications. We have now learned about different data formats, we have learned about quality control, we have done the pre-processing steps, and we have done all of those hands-on. So, you have seen how we can go through all these steps. So, we are now ready to move into the downstream analysis or the processing steps, the main processing steps and as part of that, we are going to discuss read mapping today. So, these are the concepts that we covered in this class. So, what are the challenges in read mapping? We will talk about the brute force approach, and we will talk about the evolution of this mapping algorithm. So, we will talk about BLAST and BLAT and why we need better and more efficient algorithms. So, these are the keywords that will come across throughout this presentation mapping: seeds, query, and reference. So, again, going back to the flow chart right in the NGS data analysis.

Once we have this quality control done, you can do this pre-processing if it is required, and then you move into the mapping or assembly, depending on the problem that you are working on. So, if your goal is to do signal-equal polymorphism analysis or transcriptome analysis, you will go into the read mapping side. So, you will have a reference genome against which you will map your sequence reads. If you are working with an organism where there is no reference genome, you will do the assembly right. So, today's focus will be this read mapping, where we will discuss certain challenges, etcetera, and approach and read assembly, which we will discuss in the later part of this course. So, the mapping problem is just to remind you, right? So, you have this type of read on the left, as you can see, and the goal is to identify where these reads come from in the reference genome. So, whether this comes from chromosome 1, chromosome 5, chromosome 10, etcetera, that is the goal of this mapping, right? So, for example, if we look at this schematic or this animation right, we have this read that is kind of like we found a match here in this position here in this reference sequence right and this is the mapping that is done. We can continue right with these reads, and we see that these three match there, and it is mapped there, ok, and we continue

this process right for all the reads. Finally, we get this kind of picture where we have reads all reads mapped against the reference genome and we know their location.

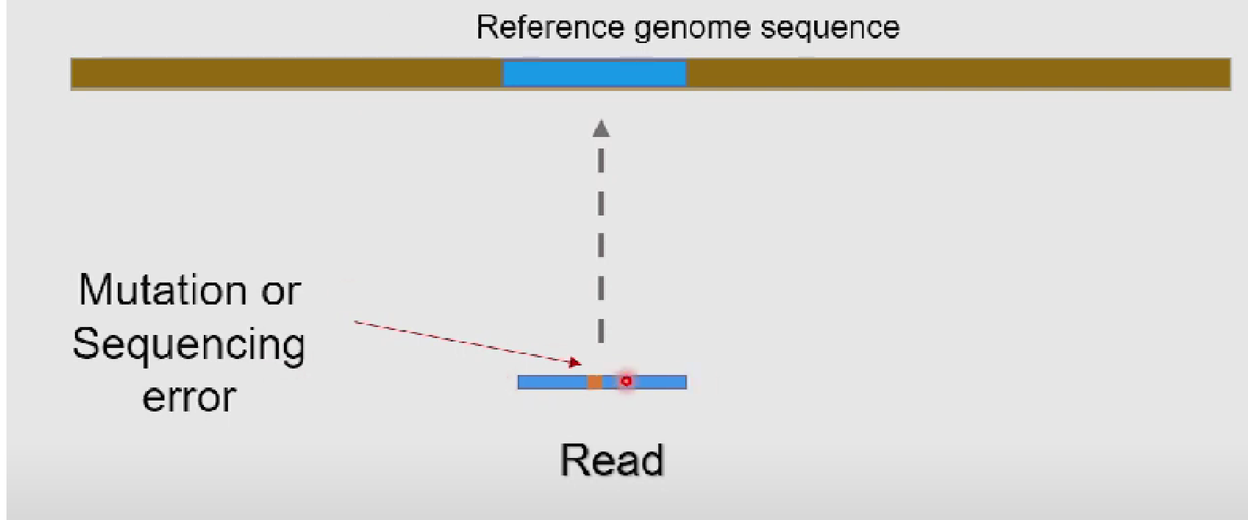




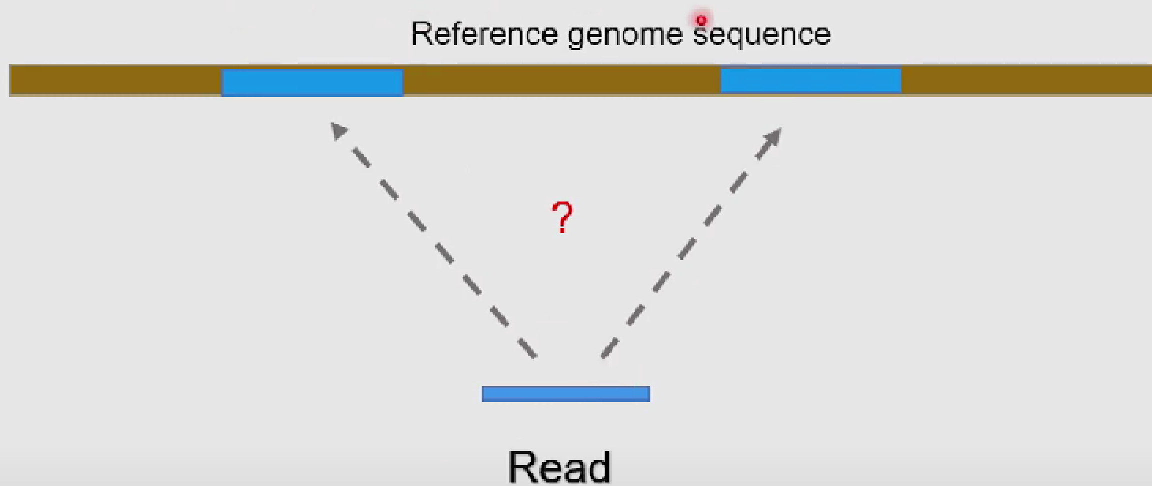
Now, this sounds very simple, but there are, of course, many challenges that we will discuss today. So, what are the major challenges here, ok? So, the first challenge is that we work with large genomes, ok? Most of the data that we will see, for example, for the human genome, will be huge, ok? So, this is like 3 into 10 to the power of 9 bases, ok? So, if you work with plants or mammalian genomes, these are really big genomes, ok? So, what is the problem with big genomes? Why is that a challenge? Because then you have many potential locations where these reads can come

from. So, you have to check all these locations, and this takes time and computational resources, ok? So, this is why the large genomes are a challenge for us. Then you have a huge number of reads. So, when we were discussing these NGS technologies, we looked at the throughput—the number of bases that we get from each experiment on those platforms—and these reads are in millions or billions. So, in today's experiments, you can get billions of reads quite easily. So, you if you can visualize in your mind that you have to repeat this process of mapping ok. For each of them ok if you have 1 billion for each of them you have to repeat this process you have to find the position in that big genome ok. So, you have this at one hand you have this big genomes right meaning many potential locations and at the other hand you have huge number of reads ok. So, if you combine them together the problem is the time requirement ok.

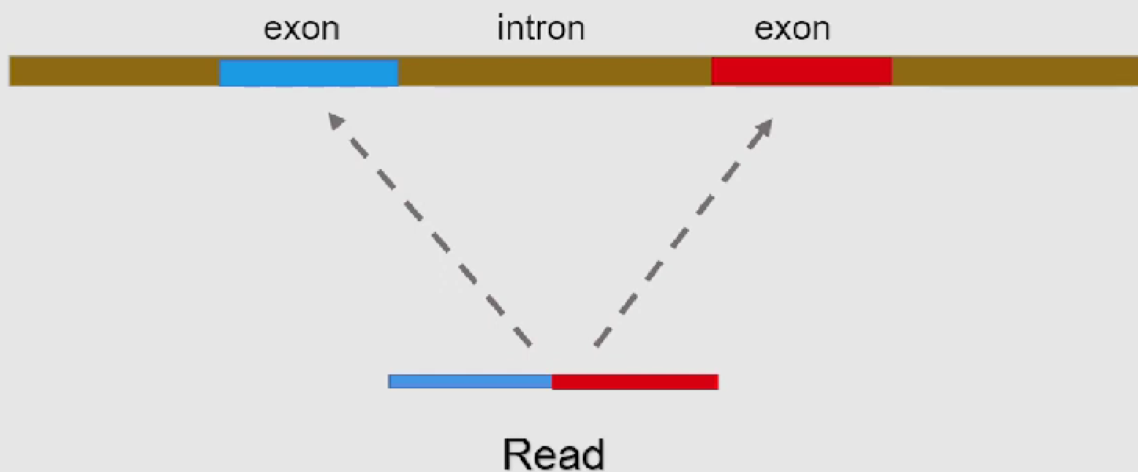
So, you will probably need a lot of time to actually complete this process ok. So, if you had only let us say few reads that you have to map against human genome then it is fine you can do it may be in an hour or 2 hours etcetera, but here we are talking about large number of reads and that actually compounds this problem ok alright. So, let us look into other challenges. So, we are dealing with mostly short reads if you are kind of working with short read data because you want to do single leaf filter polymorphism analysis, you want more accurate data, and you can also have a lot of mutations and sequencing errors in the data. So, whether you are working with short reads or long reads, this sequencing error will be there.



What does it mean if you have a mutation or sequencing error? It means you will not find a perfect match. So, you can imagine this mapping problem as a string matching problem. So, you have this read, which is kind of a combination of letters, right ATGC, and on the other hand, you have the reference sequence, which is also a combination of letters, right ATGC. So, at the end, mapping is a kind of string matching right where there is a perfect match in the genome. So, here, if you have a sequencing error, one of these bases will be changed to something else and you will see that if there is no match, you probably will not find any match in the reference genome. So, there is no perfect match. So, this is something you would have to consider while doing the mapping, right? So, any algorithm or tool that does mapping will have to consider that there may not be a perfect match, and we would have to find the best match. So, this is again a challenge. So, just to kind of represent this schematically or in an animation here, you have So, if you do not have this mutation in red, this is a perfect match, right? You find, ok, there is a perfect match, and this read comes from this position. But if you have this mutation in red, then you do not have a perfect match, and that makes this mapping problem more challenging. Then you have one more challenge, which is the repetitive regions in the genome. So, human genome or other mammalian genomes contain a lot of repeat sequences. So, one sequence repeated itself multiple times across the genome.



So, any read that comes from these regions will have multiple matches in the reference sequence, ok? So, here it is, if you can put this in a schematic figure, right here is the read, and this comes from a repetitive region that is present in two positions of the reference chromosome. Now, when you are doing this mapping, how do you decide, or can you even decide, whether this read comes from this location or this location? So, this is again what creates ambiguity here. And finally, there is another challenge. This is a specific challenge for RNA-Seq data, right? So, we will talk about this in a bit more detail when we talk about transcriptomic data analysis and read mapping in transcripts in RNA-Seq data. So, what happens in RNA when you are processing RNA samples and this mRNA you have splicing right? So, when you have these splicing events, you have these exons right that join together, and they cut out the intron in between. So, when you generate reads, it will just give you the exon sequences; it will not give you the intron sequences right because this sequencing is done on RNA, right from RNA to cDNA, and then on, that is the sequence. So, you will get these exon sequences only, not intron sequences.



Now when you go and try to map right, what it means is that you have to do something called gapped alignment. These two parts of the same read will match to two different locations in the reference genome. So, the aligner, the program that you are using, or the algorithm that you are using needs to know that this can happen because we are dealing with RNA-Seq data. So, this splicing event can happen. So, we should be aware that there could be such mappings where you will have these two different locations for one read. So, again, there are different ways to address these challenges. We will talk about this, especially this splicing in RNA-Seq data, when you go into transcriptomic data analysis. So, we have kind of understood the challenges: we are working with big genomes, we have a huge amount of data, and on top of that, we are not working with ideal data. So, we have sequencing errors, repeat sequences, etcetera. So, to make our lives more difficult, ok? Now, coming to the simple question, this is regarding the search process, right? So, this is the major component of the algorithms, ok? How do you find the read sequence location in the reference genome, and what is the search strategy, ok? So, this is what we will start with today, and we will talk about different approaches and see how these approaches work and what the limitations of each of these approaches are. So, you can start with something called a brute force approach or a trivial mapping algorithm, ok? So, what does this mean? We can simply say, "Okay, we will have the read sequence. We have the reference sequence, and we can slide that read

sequence along the reference genome sequence and see if there are certain matches around the right.

Trivial mapping algorithm

Step 1: Slide a read along the reference genome sequence and count no. of mismatches

Step 2: Set a threshold for mismatches and identify alignment/match

So, once we slide, we count the number of mismatches compared to the reference sequence, and if the number of mismatches is below a threshold, we can say this is an alignment; this is where the read maps are OK. So, this counts for the sequencing errors or mutations that might be present in the data and there might not be any perfect match, right? So, let us take some examples, and then we will understand how this works and the limitations of each of these processes. So, here is the reference sequence for example, right? This is our genome data for example, and we have the read, which is the query sequence, right? And what we want to do is find this query sequence in the reference genome, and we want to find the location of this read in the reference sequence, ok?

Reference sequence

ATCGGCATAAGCAGGCATCCAGAAAGAGGCCAACCACC

Read (Query)

CAGAATGAGCCCAAC

So, based on this brute force approach, what we do is put this read right against this reference sequence. Here, we start from the first position, right? This is the reference on top, and then query here, and we count the number of mismatches. So, the number of mismatches here is about 13, and we put the threshold as 3. So, any kind of position where you have less than 3 mismatches will say this is a good match this is an alignment, and the read maps to this position, ok. So, we can set these thresholds at different levels depending on what the expected error rate is in our sequencing data, etcetera, ok. So, clearly, you have more mismatches than the threshold. So, we cannot say this is an alignment right.

```
ATCGGCATAAGCAGGCATCCAGAAAGAGGCCAACCACC    Reference
CAGAATGAGCCCAAC                           Query
```

13 mismatches, Threshold : 3

```
ATCGGCATAAGCAGGCATCCAGAAAGAGGCCAACCACC    Reference
 CAGAATGAGCCCAAC                          Query
```

8 mismatches, Threshold : 3

```
ATCGGCATAAGCAGGCATCCAGAAAGAGGCCAACCACC    Reference
                   CAGAATGAGCCCAAC        Query
```

2 mismatches, Threshold : 3

We move along right; we are sliding along the reference sequence, and we now see again and again that the number of mismatches here is 12. Again, it is higher than the threshold.

So, we move forward and continue right here. There is a slight decrease of 8 mismatches again, but this is again higher than the threshold that we have set, and we continue this process until we reach a point where you see we have 2 mismatches and it is below the threshold that we have set. So, this is the location of this read, ok? Now, as you can see, this is a very tedious process, especially for big genomes, where you have a huge number of positions that you have to search through. And how do you also count these mismatches? You also have some algorithms that will actually count these mismatches, but these are also very slow. So, we can look at the drawbacks, right?

Drawbacks

Time requirement

Genome size 3,000,000,000 bases

Number of reads 30,000,000 (30 million)

Read length 150 bases

Number of comparisons approximately 3,000,000,000

Let's assume, time for each comparison = 1 nano second = 10^{-9} s

Total time = 3,000,000,000 × 10^{-9} × 30,000,000 seconds
= 9 × 10^{7} seconds
~ 1000 days !

So, one is the time requirement ok, the sliding window approach along with this mismatch counting right both contribute to the time required ok. So, let us take this example right, let us say genome size is about 3 billion bases, right, like the human genome. We have a number of reads, which is about 30 million, that we have generated. This is actually on the lower side because many times you will get even more right billions of reads from the sequencer, and let us consider that the read

length is about 150 bases. So, now you can imagine the scale of the problem. So, you have to check this 1 base read along this 300 3 billion base long genome, okay? So, about so, we will have about 3 billion comparisons, approximately slightly less, but it is approximately 3 billion. Now let us assume that this time for each comparison is about 1 nanosecond, ok? So, about 10 to the power minus 9 seconds, right? This is again quite optimistic. This will again depend on the computer that you are using and the system that you are using. So, let us be optimistic and say, OK, this is quite a fast process with this mismatch counting, and so, for each comparison, you have this 10 to the power minus 9 seconds. So, how much time do you require if each comparison takes about 10 to the power minus 9 seconds? It is about 3; you have 3 billion comparisons right into 10 to the power minus 9, and you have to do this for 30 million reads. So, we multiply by the number of reads here, which is 30 million. So, it comes about 9 into 10 to the power of 7 seconds, which is about 1000 days, right? So, it will take approximately 3 years to complete this mapping if there are 30 million reads. So, you can do the sequencing experiments within a week or 2 weeks, and then the mapping takes 3 years, right? That is not a good process. So, this means this will not work right. This is something we cannot utilize for 30 million for mapping NGS data, ok?

Drawbacks

- May not always find the best match
  - Alignment stops when the number of mismatches are lower than the threshold
- Finding repetitive sequences takes even more time

In addition, what you have is that you may not always find the best match. So, as we said, we have this threshold. The moment we have a match where the number of in-spatches is below that

threshold, the algorithm will stop, and this means we might not always find the best match right; there might be some match somewhere else that is even better. So, if we stop there, it will actually kind of miss the best alignment, right? So, this is one of the thresholds. Another thing that might happen is that if you want to find repetitive sequences again, if you stop after this and reach below the threshold, you will not find these repeated repeat sequences in the genome. So, multiple mappings, but if you want to find these multiple mappings, you have to scan all the positions again, which will probably take more time. Now you probably understand the drawback. So, we cannot utilize this brute force approach right now because it will take too long. So, the question is, are there alternatives? So, can we do better in certain ways? So, as we have seen, trivial mapping is very inefficient because most of the comparisons will give you a large number of mismatches. So, these are like unsuccessful attempts, right? So, we are making a lot of unsuccessful attempts, and we have just one or two successful attempts, right? So, only a few locations will show some matches, ok? Now, what we can do is maybe find some way to reduce the number of comparisons in this unsuccessful process. So, that will reduce the time, ok? So, again, if you want to compare the whole rich sequence for match or mismatch, this is also time-consuming, and it also requires memory because we utilize something called dynamic programming for mismatch checking. So, if we are using this dynamic programming using the Needleman-Wunsch and Smith-Waterman algorithms again, this takes a lot of time if you are doing this for the whole sequence. So, these are the limitations of this brute force approach, right? So, there might be some other alternatives that can do better, perhaps. One alternative that comes to mind is the last right.

So, we probably have used this BLAST for searching some sequences against databases. So, it seems quite fast, right? If you are searching one sequence, you can get this data within a few seconds or so, right? You have seen this itself, right? So, let us look at how BLAST works and whether we can apply BLAST to our problem right to mapping problem ok. So, BLAST works by something called seed and extend right and it does something called local alignment ok.

Basic Local Alignment Search Tool (BLAST)

- *Seed and extend*
- Local alignment

So, what are the steps in BLAST right? So, you have to generate seeds from the query sequence, which is the read here right. So, if you run BLAST, you will see this query sequence and subject sequence. So, here, the query sequence is the read, and the subject sequence is the reference genome. We can specify that this is the reference genome against which we want to do this BLAST, OK? So, first, you generate seeds from the query sequence.

Basic Local Alignment Search Tool (BLAST)

Steps

a) Generate seeds from the query sequence (read)

b) Generate neighboring sequences of the seeds for a given match threshold

c) Find seed hits in the reference sequence

d) Extend the seed hits until an alignment score threshold is crossed

So, we will talk about what these seeds are. These are actually substrings, only a small part of the

full sequence of the read. Then we generate neighboring sequences of the seeds for a given match threshold. So, we say we also kind of account for sequencing errors or mutations, and we kind of look at these neighboring sequences or neighborhood sequences. So, what will we take as an example to understand this better? Then we search these seeds against the reference sequence and find seed hits OK. So, we find the locations where these seeds actually hit right, and then we extend these seed hits until we reach an alignment score. Again, we can set a threshold for alignment score, and once we reach that threshold, we extend those seeds, ok? So, we will take an example to actually understand this process better, ok? Again, we come back to the same example; we have the reference and the query.

Reference sequence

ATCGGCATAAGCAGGCATCCAGAAAGAGGCCAACCACC

Read (Query)

CAGAATGAGCCCAAC

So, here the query is the read sequence, ok? So, the first step is to break down this query into seeds. So, we can choose different seed lengths, which we say k right, which we denote by k here, and we can take k equal to 5, for example, in this example here, but you can choose a different case ok? So, we are breaking down this query into seeds of length 5 right. How do you do this? So, you start with the seed, right? The first five will give you the first seed, then you have the next five, and so on. So, you can see that we can write this down. You kind of have a sliding window

and take 5 bases at a time, and that gives you the seeds, ok.

Seeding the query sequence

Query

CAGAATGAGCCCAAC

- Seed size (k) = 5 for example
- Break down query into seeds of length 5

CAGAA, AGAAT, AATGA, ATGAG, TGAGC,

GAGCC, AGCCC, GCCCA, CCCAA, CCAAC

- Generate neighborhood sequences of the seeds with 4 matches

For seed CAGAA

TAGAA CTGAA CACAA CAGTA CAGAT
AAGAA CGGAA CATAA CAGCA CAGAC
GAGAA CCGAA CAAAA CAGGA CAGAG

So, for each seed, we need to generate these neighborhood sequences, ok. So, we take one seed, right, and what are these neighborhood sequences? So, the neighborhood sequences of the seeds have four matches right. Again, we can specify whether we want 4 matches or 3 matches, depending on the sequencing error that we want to accommodate in our analysis. So, for seed CAGA, we look at the neighboring sequences with 4 matches, which means that 4 bases will be

identical to the seed and only one base can vary. So, here are all the neighbors of this seed, ok? Now what it means now is that we are accounting for this sequencing error at different locations of the seed, ok? So, in the next step, we search these seeds and the neighborhood sequences in the reference sequence, and since we are considering all the variations, there should be a perfect match, ok?

Search reference sequence

- Search seeds and neighborhood sequences in the reference sequence
- There should be a perfect match!

Reference sequence

ATCGGCATAAGCAGGCATCCAGAAAGAGGCCAACCACC
CAGAA

And again, if you kind of slide across this reference sequence, you will see at this position you have this match. Now you might ask, How is it different from the earlier brute force approach? So, here we are taking actually a small part of the sequence, and this actually finds this and can check for match only, right? You do not, and we are not looking for mismatches. So, we do not need to use something like dynamic programming or more time-consuming algorithms because we are looking for a perfect match. We can simply kind of look at identity, and so that process is much faster. But we still have this sliding thing right; we are kind of looking at linearly across this reference sequence.

- Extend until a threshold alignment score S is reached
- Stop if the alignment score falls below a threshold T

Reference sequence

ATCGGCATAAGCAGGCATCCAGAAAGAGGCCAACCACC

← CCCAGAAAG →

So, once we find this match, we kind of extend in both directions, and we extend until we cross this alignment score S. So, if we cross this threshold score S, then we say we have found some match, right? We have mapped the seed. And then we stop, right? When we stop after we have crossed the S, we stop if we see that the alignment score is falling below a threshold, ok? So, we can define these S and T depending on our requirements. So, this is the whole thing, right? So, the idea is that instead of matching the full query sequence into the reference right, we are taking a part of this query sequence and mapping it against the reference right. So, this kind of speeds up the matching or mismatching process.

Now, the drawbacks are right. So, as I said, there might be some improvement in this match mismatch step, right? So, again, take going back to the same example. We are looking at the same data: we have 3 billion bases in the reference sequence, we have 30 million reads, and we have a similar number of comparisons. But let us assume that the time for each comparison now is 10 to the power minus 10 seconds. So, we have improved there, right? So, we are doing that faster. So, what you do mean is that the total time that will be required right now will actually be less, but still, it is quite high right now; it is about 100 days, ok? So, it is still not acceptable right now; it takes about 3–4 months, which is not acceptable for us. Can we do better? Is there any way we can improve on this? So, the question is, how do you actually improve on this? Now, one of the things you probably have noticed now is that whether we are going with the brute force approach or with

blast, we are kind of linearly searching through the reference sequence, and we are kind of sliding through the reference sequence. Instead of that, can we also seed or index the reference sequence? Can we create some of the fragments of the reference sequence right? We can take out parts of the reference sequence right and then search against the query right. So, why is it better? Why is it a better idea than what we have done so far? It is because our query sequence is smaller than the reference sequence. So, if you are searching linearly through the reference sequence, it takes more time to search a single seed or single sequence, but if you do the opposite, if we create those seeds from the reference sequence and search against the read sequence, this will be faster because the read sequence is smaller in length, ok? So, we may have to search more seeds, but the process of comparison for each seed is much faster. So, this is kind of the idea and there is a tool that kind of does this is called blast like alignment tool or BLAT ok.

Blast-like alignment tool (BLAT)

Steps

- Seed the reference sequence
- Then search through the query sequence

Reference sequence

ATCGGCATAAGCAGGCATCCAGAAAGAGGCCAACCACC

Read (Query)

CAGAATGAGCCCAAC

Reference sequence

ATCGGCATAAGCAGGCATCCAGAAAGAGGCCAACCACC

Seed size (k)= 5

Generate seeds of length 5 from reference

ATCGG, CGGCA, GGCAT, etc..

So, how does it work it seeds the reference sequence ok. So, these are the steps so it sees the reference sequence and then it searches through the query sequence ok. So, again we go back to the example right where we have this reference sequence and we have the read sequence or the query sequence and we take the same seed size and we generate seeds of length 5 from reference ok. So, again we follow the same process right we start from the beginning we take seeds of length 5 right we start we count only 5 bases that is a seed then we move on right to the next base take next 5 and move on right so on and these are the seeds that we will get right as we move

along right you can do this for the whole reference sequence ok. Now once you have found this right you can also generate neighborhood sequence of the seeds considering all the variations and then you can search through the query sequence ok. Now, it is the reverse right compared to what we have discussed now before right blast.

BLAT

- Generate neighborhood sequence of the seeds
- Search through the query sequence – find hits

```
                 CAGAA
Read (Query)     CAGAATGAGCCCAAC
```

So, the whole thing, the whole process has reversed ok. So, without going into all the details, because we have seen how you generate these neighborhood sequences you generate those for the seeds here, and then we search through the query sequence something like this, and then you find hits ok. Again you expect perfect matches because we have considered these variations. Right when you are considering these neighborhood sequences with certain matches, we are considering sequencing errors. So, you should find perfect matches, and you do not have to apply dynamic programming or something else to actually find the number of mismatches, etcetera, right? So, once you find this hit, you just extend again until a threshold alignment score is reached, ok? So, this is something that you might want to see to see whether this actually works and whether we

actually improve on time.

BLAT

- Extend till a threshold alignment score S

CAGAAT →

Read (Query) CAGAATGAGCCCAAC

Drawbacks

Time requirement

Genome size 3,000,000,000 bases

Number of reads 30,000,000 (30 million)

Read length 150 bases

Number of comparisons approximately 3,000,000,000

Let's assume, time for each comparison = 10^{-11} s

More improvement here!

Total time = 3,000,000,000 × 10^{-11} × 30,000,000 seconds
= 9 × 10^{5} seconds
~ 10 days !

We can actually look at that time calculation again, and maybe we can say we have improved upon it a lot more. So, now, every comparison takes even less time: 10 to the power minus 11 seconds, and you can calculate the total time, which is still about 10 days. So, this is something again that is probably not enough, ok? For 30 million reach, if it takes 10 days, if you increase the number of reach, if you are processing millions of reach, it will take even longer. So, if you are processing 300 billion, it will take 100 days. So, this will increase the time, and that is still not

acceptable, right? We want to do this within a few hours, maybe a day or two, not 10 days for 30 million. So, what it means is that we need a more efficient algorithm. So, whatever we have discussed, the brute force approach plus plus they are not good enough for this problem, ok. And this is because the number of reads that we get from the data is often more than 30 right billions and will require a lot more time, ok.

Need more efficient algorithms

- Number of reads are often more than 30 million
- Time required will be more
- Majority of comparisons are not successful
- Can we further reduce the number of such comparisons?

And one of the problems here with all these algorithms is that the majority of comparisons are not successful comparisons, ok. So, we are doing these comparisons, and most of them will not give any positive results. So, we are spending a lot of time doing these unsuccessful comparisons. So, can we reduce the number of such comparisons? So, if we can reduce this number of comparisons, then we can improve on time. So, we can probably focus on the successful comparisons, and we can get to the results much quicker, ok? So, these are the references that we have talked about today. And to summarize, we have seen that if you want to map a large number of reads, this is quite a challenging problem that requires a lot of time and probably computational resources. So, this is something we would have to think about carefully, right? There are two important considerations, one of which is time. You do not want algorithms that take too much time because

that is not what you want because you will be done with sequencing within a few days or a week. And if your program takes a few weeks, that is not good enough, ok? And also, you do not want to use too much memory. This is again very important because if you can run this analysis on your desktop system, that will be great, but if you cannot, if it takes a lot of memory, if you need sophisticated servers with large computational resources that will again limit the accessibility. So, these are two important considerations. What you have also seen is that blast is too slow right? So, we cannot, of course, use the brute force approach at all; that will probably take forever, but even blast that people used for searching through databases; this is also too slow for mapping when you have this large number of reads; this number approaches billions; this process takes too long. We have seen an improvement over blast, which is that blast takes less time than blast, but it is still inefficient if you are going with billions of reads, ok? And this means we need more efficient mapping algorithms, ok? And this is something that has developed with the advancement of sequencing technology. As people have seen, we are getting more throughput from the sequencers. There was a need for more efficient mapping algorithms, and that is why we will see that in the next class there will be a plethora of algorithms that can do this job in a much more efficient manner. So, we will discuss these algorithms over the next few classes, and we will then look into the best one. We will do a comparison, we will look into the best algorithm, and then we will also discuss the tools that can do this very efficiently. Thank you.