BLAST & Genome assembly

Solon P. Pissis    Tomáš Flouri

Heidelberg Institute for Theoretical Studies

November 17, 2012
1 Introduction
   - Introduction

2 BLAST
   - What is BLAST?
   - The algorithm

3 Genome assembly
   - De novo assembly
   - Mapping assembly

4 Conclusion
   - Overview
Contents

1. Introduction
2. BLAST
3. Genome assembly
4. Conclusion
Introduction

Sequence alignment is the process of comparing two or more strings of letters (e.g. nucleotides or amino acids) to infer their similarity.
Sequence alignment is the process of comparing two or more strings of letters (e.g. nucleotides or amino acids) to infer their similarity.

Pairwise sequence alignment is the process of comparing only two strings.
Sequence alignment is the process of comparing two or more strings of letters (e.g. nucleotides or amino acids) to infer their similarity.

Pairwise sequence alignment is the process of comparing only two strings.

Useful in dozens of biological applications (SSE- and GPU-based accelerated implementations).
Sequence alignment is the process of comparing two or more strings of letters (e.g. nucleotides or amino acids) to infer their similarity.

Pairwise sequence alignment is the process of comparing only two strings.

Useful in dozens of biological applications (SSE- and GPU-based accelerated implementations).

**BLAST:** Basic Local Alignment Search Tool is a set of programs for fast approximate comparison of biological sequences, such as the amino-acid sequences of different proteins or the nucleotides of DNA sequences.
Sequence alignment is the process of comparing two or more strings of letters (e.g. nucleotides or amino acids) to infer their similarity.

Pairwise sequence alignment is the process of comparing only two strings.

Useful in dozens of biological applications (SSE- and GPU-based accelerated implementations).

**BLAST**: Basic Local Alignment Search Tool is a set of programs for fast approximate comparison of biological sequences, such as the amino-acid sequences of different proteins or the nucleotides of DNA sequences.

**Genome assembly**: taking a huge number of DNA sequences and putting them back together to create a representation of the genome from which the DNA originated.
What is BLAST?

BLAST: a set of programs

- Basic Local Alignment Search Tool is a set of programs for fast and approximate comparison of biological sequences.
What is BLAST?

**BLAST: a set of programs**

- Basic Local Alignment Search Tool is a set of programs for fast and approximate comparison of biological sequences.
- In particular, BLAST is useful for the comparison between a *query sequence* and a *library or database of sequences*, in order to identify library sequences that resemble the query sequence above a certain threshold.
BLAST: a set of programs

- Basic Local Alignment Search Tool is a set of programs for fast and approximate comparison of biological sequences.
- In particular, BLAST is useful for the comparison between a query sequence and a library or database of sequences, in order to identify library sequences that resemble the query sequence above a certain threshold.
- The five traditional BLAST implementations are:
BLAST: a set of programs

- Basic Local Alignment Search Tool is a set of programs for fast and approximate comparison of biological sequences.
- In particular, BLAST is useful for the comparison between a query sequence and a library or database of sequences, in order to identify library sequences that resemble the query sequence above a certain threshold.
- The five traditional BLAST implementations are:
  - BLASTN: both the database and the query are nucleotide sequences
What is BLAST?

**BLAST: a set of programs**

- Basic Local Alignment Search Tool is a set of programs for fast and approximate comparison of biological sequences.
- In particular, BLAST is useful for the comparison between a *query sequence* and a *library or database of sequences*, in order to identify library sequences that resemble the query sequence above a certain threshold.
- The five traditional BLAST implementations are:
  - BLASTN: both the database and the query are nucleotide sequences
  - BLASTP: both the database and the query are protein sequences
**What is BLAST?**

**BLAST: a set of programs**

- Basic Local Alignment Search Tool is a set of programs for fast and approximate comparison of biological sequences.
- In particular, BLAST is useful for the comparison between a *query sequence* and a *library or database of sequences*, in order to identify library sequences that resemble the query sequence above a certain threshold.
- The five traditional BLAST implementations are:
  - BLASTN: both the database and the query are nucleotide sequences
  - BLASTP: both the database and the query are protein sequences
  - BLASTX: the database are protein sequences and the query is nucleotide translated into protein sequence
What is BLAST?

BLAST: a set of programs

- Basic Local Alignment Search Tool is a set of programs for fast and approximate comparison of biological sequences.
- In particular, BLAST is useful for the comparison between a query sequence and a library or database of sequences, in order to identify library sequences that resemble the query sequence above a certain threshold.
- The five traditional BLAST implementations are:
  - BLASTN: both the database and the query are nucleotide sequences
  - BLASTP: both the database and the query are protein sequences
  - BLASTX: the database are protein sequences and the query is nucleotide translated into protein sequence
  - TBLASTN: the database are nucleotide translated into protein sequence and the query is a protein sequence
BLAST: a set of programs

- Basic Local Alignment Search Tool is a set of programs for fast and approximate comparison of biological sequences.
- In particular, BLAST is useful for the comparison between a query sequence and a library or database of sequences, in order to identify library sequences that resemble the query sequence above a certain threshold.
- The five traditional BLAST implementations are:
  - BLASTN: both the database and the query are nucleotide sequences
  - BLASTP: both the database and the query are protein sequences
  - BLASTX: the database are protein sequences and the query is nucleotide translated into protein sequence
  - TBLASTN: the database are nucleotide translated into protein sequence and the query is a protein sequence
  - TBLASTX: both the database and the query are nucleotide translated into protein sequences
The algorithm

- “Why not Smith-Waterman algorithm?”
The algorithm

- “Why not Smith-Waterman algorithm?”
- Smith-Waterman algorithm computes the optimal (maximum scoring) local alignment between two sequences.
The algorithm

- “Why not Smith-Waterman algorithm?”
- Smith-Waterman algorithm computes the optimal (maximum scoring) local alignment between two sequences.
- In biological applications, we usually need to infer the statistically significant alignments very fast.
The algorithm

- “Why not Smith-Waterman algorithm?”
- Smith-Waterman algorithm computes the optimal (maximum scoring) local alignment between two sequences.
- In biological applications, we usually need to infer the statistically significant alignments very fast.
- BLAST does not explore the entire search space (DP matrix) but it minimizes the search space for efficiency...
The algorithm

- “Why not Smith-Waterman algorithm?”
- Smith-Waterman algorithm computes the optimal (maximum scoring) local alignment between two sequences.
- In biological applications, we usually need to infer the statistically significant alignments very fast.
- BLAST does not explore the entire search space (DP matrix) but it minimizes the search space for efficiency...
- ...at the cost of sensitivity
The algorithm

- “Why not Smith-Waterman algorithm?”
- Smith-Waterman algorithm computes the optimal (maximum scoring) local alignment between two sequences.
- In biological applications, we usually need to infer the statistically significant alignments very fast.
- BLAST does not explore the entire search space (DP matrix) but it minimizes the search space for efficiency...
- ...at the cost of sensitivity
- It uses three layers of rules to sequentially identify refine potential high scoring pairs (HSPs).
The algorithm

- “Why not Smith-Waterman algorithm?”
- Smith-Waterman algorithm computes the optimal (maximum scoring) local alignment between two sequences.
- In biological applications, we usually need to infer the statistically significant alignments very fast.
- BLAST does not explore the entire search space (DP matrix) but it minimizes the search space for efficiency...
- ...at the cost of sensitivity
- It uses three layers of rules to sequentially identify refine potential high scoring pairs (HSPs).
- These heuristics layers—seeding, extension, and evaluation—form a stepwise refinement procedure.
The algorithm

- “Why not Smith-Waterman algorithm?”
- Smith-Waterman algorithm computes the optimal (maximum scoring) local alignment between two sequences.
- In biological applications, we usually need to infer the statistically significant alignments very fast.
- BLAST does not explore the entire search space (DP matrix) but it minimizes the search space for efficiency...
- ...at the cost of sensitivity
- It uses three layers of rules to sequentially identify refine potential high scoring pairs (HSPs).
- These **heuristics** layers—seeding, extension, and evaluation—form a stepwise refinement procedure.
- Allows for sampling the entire search space without wasting time on dissimilar regions.
BLAST assumes that significant alignments have common subwords (substrings or factors) of a fixed-length $W$. 
The algorithm

The algorithm: seeding

- BLAST assumes that significant alignments have common subwords (substrings or factors) of a fixed-length $W$.
- It first determines the locations of all the common exact matching substrings which are called *word hits*. 
The algorithm: seeding

- BLAST assumes that significant alignments have common subwords (substrings or factors) of a fixed-length $W$.
- It first determines the locations of all the common exact matching substrings which are called word hits.
- Only those regions with word hits will be used as alignment seeds.
The algorithm: seeding

- BLAST assumes that significant alignments have common subwords (substrings or factors) of a fixed-length $W$.
- It first determines the locations of all the common exact matching substrings which are called word hits.
- Only those regions with word hits will be used as alignment seeds.
- In this way BLAST ignores a large fraction of search space.
The algorithm: seeding

- BLAST assumes that significant alignments have common *subwords* (substrings or factors) of a fixed-length $W$.
- It first determines the locations of all the common exact matching substrings which are called *word hits*.
- Only those regions with word hits will be used as alignment *seeds*.
- In this way BLAST ignores a large fraction of search space.
- The *neighborhood* of a subword contains the word itself and all other words whose score is $\leq T$ when compared via the *substitution matrix* to the subword.
The algorithm: seeding

- BLAST assumes that significant alignments have common *subwords* (substrings or factors) of a fixed-length $W$.
- It first determines the locations of all the common exact matching substrings which are called *word hits*.
- Only those regions with word hits will be used as alignment *seeds*.
- In this way BLAST ignores a large fraction of search space.
- The *neighborhood* of a subword contains the word itself and all other words whose score is $\leq T$ when compared via the *substitution matrix* to the subword.
- We may adjust $T$ to control the size of the neighborhood—affecting speed and sensitivity.
The algorithm: seeding

- BLAST assumes that significant alignments have common subwords (substrings or factors) of a fixed-length $W$.
- It first determines the locations of all the common exact matching substrings which are called word hits.
- Only those regions with word hits will be used as alignment seeds.
- In this way BLAST ignores a large fraction of search space.
- The neighborhood of a subword contains the word itself and all other words whose score is $\leq T$ when compared via the substitution matrix to the subword.
- We may adjust $T$ to control the size of the neighborhood—affecting speed and sensitivity.
- Hence, the interplay between $W$, $T$, and the substitution matrix is critical!!!
The algorithm: extension

Cumulative score

hit extension

A D H W R ...
A E H S Q ...

X
The algorithm: extension

- Once the search space is seeded, alignments can be generated by starting from the individual seeds.
The algorithm: extension

- Once the search space is seeded, alignments can be generated by starting from the individual seeds.
- BLAST extends a longer alignment between the query and the database sequence in the left and right direction of the word.
The algorithm: extension

- Once the search space is seeded, alignments can be generated by starting from the individual seeds.
- BLAST extends a longer alignment between the query and the database sequence in the left and right direction of the word.
- It only searches a subset of the space, so it needs a mechanism to know when to stop the extension procedure.
The algorithm: extension

- Once the search space is seeded, alignments can be generated by starting from the individual seeds.
- BLAST extends a longer alignment between the query and the database sequence in the left and right direction of the word.
- It only searches a subset of the space, so it needs a mechanism to know when to stop the extension procedure.
- It uses a threshold $X$ representing how much the score is allowed to drop off since the last maximum.
The algorithm: extension

- Once the search space is seeded, alignments can be generated by starting from the individual seeds.
- BLAST extends a longer alignment between the query and the database sequence in the left and right direction of the word.
- It only searches a subset of the space, so it needs a mechanism to know when to stop the extension procedure.
- It uses a threshold $X$ representing how much the score is allowed to drop off since the last maximum.
- The extension is stopped as soon as the sum score decreases by more than $X$ when compared with the highest value obtained during the extension process.
The algorithm: extension

- Once the search space is seeded, alignments can be generated by starting from the individual seeds.
- BLAST extends a longer alignment between the query and the database sequence in the left and right direction of the word.
- It only searches a subset of the space, so it needs a mechanism to know when to stop the extension procedure.
- It uses a threshold $X$ representing how much the score is allowed to drop off since the last maximum.
- The extension is stopped as soon as the sum score decreases by more than $X$ when compared with the highest value obtained during the extension process.
- The alignment is trimmed back to the maximum score.
The algorithm: extension

- Once the search space is seeded, alignments can be generated by starting from the individual seeds.
- BLAST extends a longer alignment between the query and the database sequence in the left and right direction of the word.
- It only searches a subset of the space, so it needs a mechanism to know when to stop the extension procedure.
- It uses a threshold $X$ representing how much the score is allowed to drop off since the last maximum.
- The extension is stopped as soon as the sum score decreases by more than $X$ when compared with the highest value obtained during the extension process.
- The alignment is trimmed back to the maximum score.
- It is generally a good idea to use a large value for $X$, which reduces the risk of premature termination.
Once seeds have been extended in both directions to create alignments, these alignments are *evaluated* (post-processed) to determine if they are statistically significant.
The algorithm: evaluation

- Once seeds have been extended in both directions to create alignments, these alignments are evaluated (post-processed) to determine if they are statistically significant.
- The significant alignments are termed HSPs (High Scoring Pairs).
The algorithm: evaluation

- Once seeds have been extended in both directions to create alignments, these alignments are *evaluated* (post-processed) to determine if they are statistically significant.
- The significant alignments are termed HSPs (High Scoring Pairs).
- At the simplest level we can use an optional alignment score threshold (cut-off) $S$—empirically determined—to sort the alignments into low and high scoring.
Once seeds have been extended in both directions to create alignments, these alignments are evaluated (post-processed) to determine if they are statistically significant.

The significant alignments are termed HSPs (High Scoring Pairs).

At the simplest level we can use an optional alignment score threshold (cut-off) $S$—empirically determined—to sort the alignments into low and high scoring.

By examining the distribution of the alignment scores modeled by comparing random sequences, $S$ can be determined such that its value is large enough to guarantee the significance of the remaining HSPs.
The algorithm: evaluation

- BLAST next assesses the statistical significance of each HSP score by using a *final threshold*. 
The algorithm: evaluation

- BLAST next assesses the statistical significance of each HSP score by using a *final threshold*.
- It computes the probability $p$ of observing a score $S$ equal to or greater than score $x$ by exploiting the *Gumbel extreme value distribution* (GEDV).
The algorithm: evaluation

- BLAST next assesses the statistical significance of each HSP score by using a *final threshold*.
- It computes the probability $p$ of observing a score $S$ equal to or greater than score $x$ by exploiting the *Gumbel extreme value distribution* (GEDV).
- It is shown that the distribution of Smith-Waterman local alignment scores between two random sequences follows GEDV.
The algorithm: evaluation

- BLAST next assesses the statistical significance of each HSP score by using a *final threshold*.
- It computes the probability $p$ of observing a score $S$ equal to or grater than score $x$ by exploiting the *Gumbel extreme value distribution* (GEDV).
- It is shown that the distribution of Smith-Waterman local alignment scores between two random sequences follows GEDV.
- The computation of $p$ is based on statistical parameters depending upon the substitution matrix, the gap penalties, and the problem size.
The algorithm: evaluation

- BLAST next assesses the statistical significance of each HSP score by using a *final threshold*.
- It computes the probability $p$ of observing a score $S$ equal to or greater than score $x$ by exploiting the *Gumbel extreme value distribution* (GEDV).
- It is shown that the distribution of Smith-Waterman local alignment scores between two random sequences follows GEDV.
- The computation of $p$ is based on statistical parameters depending upon the substitution matrix, the gap penalties, and the problem size.
- The final threshold $E$ (computed by $p$) of a database match is the number of times that a random sequence would obtain a score $S$ higher than $x$ by chance.
Contents

1 Introduction
2 BLAST
3 Genome assembly
4 Conclusion
Genome assembly

- *Genome assembly* is the process of taking a huge number of DNA sequences and putting them back together to create a representation of the genome from which the DNA originated.
Genome assembly

- *Genome assembly* is the process of taking a huge number of DNA sequences and putting them back together to create a representation of the genome from which the DNA originated.

- *De novo*: assembling short reads to create full-length—sometimes novel—sequences.
Genome assembly

- **Genome assembly** is the process of taking a huge number of DNA sequences and putting them back together to create a representation of the genome from which the DNA originated.

- **De novo**: assembling short reads to create full-length—sometimes novel—sequences.

- **Mapping**: assembling reads by aligning them against an existing reference sequence—building a sequence that is similar but not necessarily identical to the reference.
Genome assembly

- *Genome assembly* is the process of taking a huge number of DNA sequences and putting them back together to create a representation of the genome from which the DNA originated.

- *De novo*: assembling short reads to create full-length—sometimes novel—sequences.

- *Mapping*: assembling reads by aligning them against an existing reference sequence—building a sequence that is similar but not necessarily identical to the reference.

- Genome assembly is generally a very difficult computational problem, and since 2005, probably, one of the hottests in Bioinformatics.
**Genome assembly**

- *Genome assembly* is the process of taking a huge number of DNA sequences and putting them back together to create a representation of the genome from which the DNA originated.

- *De novo*: assembling short reads to create full-length—sometimes novel—sequences.

- *Mapping*: assembling reads by aligning them against an existing reference sequence—building a sequence that is similar but not necessarily identical to the reference.

- Genome assembly is generally a very difficult computational problem, and since 2005, probably, one of the hottest in Bioinformatics.

- In terms of time and space complexity, de novo assembly is orders of magnitude slower and more memory intensive than mapping assembly.
Genome assembly: DNA sequencing

ATTAGCATAC...
DNA sequencing includes several methods and technologies that are used for determining the exact order of the nucleotide bases—adenine, guanine, cytosine, and thymine—in a DNA macromolecule.
DNA sequencing includes several methods and technologies that are used for determining the exact order of the nucleotide bases—adenine, guanine, cytosine, and thymine—in a DNA macromolecule.

The traditional sequencing methods, named after Sanger and developed in the mid 70’s, had been the workhorse technology for DNA sequencing for almost thirty years.
DNA sequencing includes several methods and technologies that are used for determining the exact order of the nucleotide bases—adenine, guanine, cytosine, and thymine—in a DNA macromolecule.

The traditional sequencing methods, named after Sanger and developed in the mid 70’s, had been the workhorse technology for DNA sequencing for almost thirty years.

With the paramount goal of analysing the human genome, the throughput demand of DNA sequencing increased by an unexpected magnitude, leading to new developments.
DNA sequencing includes several methods and technologies that are used for determining the exact order of the nucleotide bases—adenine, guanine, cytosine, and thymine—in a DNA macromolecule.

The traditional sequencing methods, named after Sanger and developed in the mid 70’s, had been the workhorse technology for DNA sequencing for almost thirty years.

With the paramount goal of analysing the human genome, the throughput demand of DNA sequencing increased by an unexpected magnitude, leading to new developments.

The speed, accuracy, efficiency, and cost-effectiveness of sequencing technology have been improving since.
In 2005: the milestone publication of the sequencing-by-synthesis (SBS) technology (Margulies et al., 2005), and the multiplex polony sequencing protocol of George Church’s laboratory (Shendure et al., 2005).
Genome assembly: Next-generation sequencing

- In 2005: the milestone publication of the sequencing-by-synthesis (SBS) technology (Margulies et al., 2005), and the multiplex polony sequencing protocol of George Church’s laboratory (Shendure et al., 2005).
- Short sequences (reads) of length 25-100 base pairs (bp), which after sixteen months on the market had increased to 250 bp.
Genome assembly: Next-generation sequencing

- In 2005: the milestone publication of the sequencing-by-synthesis (SBS) technology (Margulies et al., 2005), and the multiplex polony sequencing protocol of George Church’s laboratory (Shendure et al., 2005).
- Short sequences (reads) of length 25-100 base pairs (bp), which after sixteen months on the market had increased to 250 bp.
- Recent advances have raised the mark again to more than 500 bp—drawing near today’s Sanger sequencing read length of 750 bp.
Introduction

BLAST

Genome assembly

Conclusion

Genome assembly: Next-generation sequencing

- In 2005: the milestone publication of the sequencing-by-synthesis (SBS) technology (Margulies et al., 2005), and the multiplex polony sequencing protocol of George Church’s laboratory (Shendure et al., 2005).
- Short sequences (reads) of length 25-100 base pairs (bp), which after sixteen months on the market had increased to 250 bp.
- Recent advances have raised the mark again to more than 500 bp—drawing near today’s Sanger sequencing read length of 750 bp.
- Apart from read length, the massive amount (tens of millions) of sequencing reads that can be produced in a single instrument run for a given cost is another important aspect.
- These advances is what we call next-generation sequencing (NGS).
Genome assembly: Impact

- The impact that these next-generation sequencing innovations will have in clinical genetics will certainly be crucial.
The impact that these next-generation sequencing innovations will have in clinical genetics will certainly be crucial.

The low-scale, targeted gene/mutation analysis currently dominating clinical genetics will ultimately be replaced by large-scale sequencing of entire disease gene pathways and networks.
The impact that these next-generation sequencing innovations will have in clinical genetics will certainly be crucial.

The low-scale, targeted gene/mutation analysis currently dominating clinical genetics will ultimately be replaced by large-scale sequencing of entire disease gene pathways and networks.

Eventually, the perceived clinical benefit of whole-genome sequencing will outweigh the cost of the procedure.
The impact that these next-generation sequencing innovations will have in clinical genetics will certainly be crucial.

The low-scale, targeted gene/mutation analysis currently dominating clinical genetics will ultimately be replaced by large-scale sequencing of entire disease gene pathways and networks.

Eventually, the perceived clinical benefit of whole-genome sequencing will outweigh the cost of the procedure.

Allowing for these tests to be performed on a routine basis for diagnostic purposes.
The impact that these next-generation sequencing innovations will have in clinical genetics will certainly be crucial.

The low-scale, targeted gene/mutation analysis currently dominating clinical genetics will ultimately be replaced by large-scale sequencing of entire disease gene pathways and networks.

Eventually, the perceived clinical benefit of whole-genome sequencing will outweigh the cost of the procedure.

Allowing for these tests to be performed on a routine basis for diagnostic purposes.

Or perhaps in the form of a screening programme, that could be used to guide personalised medical treatments throughout the lifetime of the individual.
Genome assembly: Impact

- The impact that these next-generation sequencing innovations will have in clinical genetics will certainly be crucial.
- The low-scale, targeted gene/mutation analysis currently dominating clinical genetics will ultimately be replaced by large-scale sequencing of entire disease gene pathways and networks.
- Eventually, the perceived clinical benefit of whole-genome sequencing will outweigh the cost of the procedure.
- Allowing for these tests to be performed on a routine basis for diagnostic purposes.
- Or perhaps in the form of a screening programme, that could be used to guide personalised medical treatments throughout the lifetime of the individual.
- 2M characterized species of plants and animals—not accounting for microbes; only 3791 completed genomes.
De novo assembly: what is it?

1. Generate reads
2. Find overlapping reads
3. Assemble reads into contigs
4. Join contigs into scaffolds using mate pairs
5. Join scaffolds into “finished” sequence
De novo assembly: what is it?

- *De novo* assembly is a hierarchical data structure that maps the sequence data to a putative reconstruction of the target.
**De novo assembly: what is it?**

- *De novo* assembly is a hierarchical data structure that maps the sequence data to a putative reconstruction of the target.
- It groups reads into *contigs* and contigs into *scaffolds*.
**De novo assembly: what is it?**

- *De novo* assembly is a hierarchical data structure that maps the sequence data to a putative reconstruction of the target.
- It groups reads into *contigs* and contigs into *scaffolds*.
- Contigs provide a multiple sequence alignment of reads plus the consensus sequence.
De novo assembly: what is it?

- De novo assembly is a hierarchical data structure that maps the sequence data to a putative reconstruction of the target.
- It groups reads into contigs and contigs into scaffolds.
- Contigs provide a multiple sequence alignment of reads plus the consensus sequence.
- Scaffolds define the contig order and orientation and the sizes of the gaps between contigs using mate pairs (paired-end) information.
**De novo assembly: what is it?**

- *De novo* assembly is a hierarchical data structure that maps the sequence data to a putative reconstruction of the target.
- It groups reads into *contigs* and contigs into *scaffolds*.
- Contigs provide a multiple sequence alignment of reads plus the consensus sequence.
- Scaffolds define the contig order and orientation and the sizes of the gaps between contigs using mate pairs (paired-end) information.
- Assemblies are measured by the size and accuracy of their contigs and scaffolds.
**De novo assembly: what is it?**

- *De novo* assembly is a hierarchical data structure that maps the sequence data to a putative reconstruction of the target.
- It groups reads into *contigs* and contigs into *scaffolds*.
- Contigs provide a multiple sequence alignment of reads plus the consensus sequence.
- Scaffolds define the contig order and orientation and the sizes of the gaps between contigs using mate pairs (paired-end) information.
- Assemblies are measured by the size and accuracy of their contigs and scaffolds.
- Assembling a genome using *many short* NGS reads requires a different approach than the methods developed for the *fewer but longer* reads produced by Sanger sequencing.
De novo assembly: what is it?

- De novo assembly is a hierarchical data structure that maps the sequence data to a putative reconstruction of the target.
- It groups reads into contigs and contigs into scaffolds.
- Contigs provide a multiple sequence alignment of reads plus the consensus sequence.
- Scaffolds define the contig order and orientation and the sizes of the gaps between contigs using mate pairs (paired-end) information.
- Assemblies are measured by the size and accuracy of their contigs and scaffolds.
- Assembling a genome using many short NGS reads requires a different approach than the methods developed for the fewer but longer reads produced by Sanger sequencing.
- There are two basic algorithmic approaches for de novo assembly: overlap graphs and de Bruijn graphs.
De novo assembly algorithms: Overlap graphs

**Figure:** Colored nucleotides indicate overlaps between reads
De novo assembly algorithms: Overlap graphs

- Compute all pair-wise overlaps between the reads and capture this information in a graph.
De novo assembly algorithms: Overlap graphs

- Compute all pair-wise overlaps between the reads and capture this information in a graph.
- Each node in the graph corresponds to a read, and an edge denotes an overlap between two reads.
De novo assembly algorithms: Overlap graphs

- Compute all pair-wise overlaps between the reads and capture this information in a graph.
- Each node in the graph corresponds to a read, and an edge denotes an overlap between two reads.
- The overlap graph is used to compute an arrangement of reads and a consensus sequence of contigs.
De novo assembly algorithms: Overlap graphs

- Compute all pair-wise overlaps between the reads and capture this information in a graph.
- Each node in the graph corresponds to a read, and an edge denotes an overlap between two reads.
- The overlap graph is used to compute an arrangement of reads and a consensus sequence of contigs.
- This method works best when there is a small number of reads with significant overlap.
De novo assembly algorithms: Overlap graphs

- Compute all pair-wise overlaps between the reads and capture this information in a graph.
- Each node in the graph corresponds to a read, and an edge denotes an overlap between two reads.
- The overlap graph is used to compute an arrangement of reads and a consensus sequence of contigs.
- This method works best when there is a small number of reads with significant overlap.
- Some NGS assemblers use overlap graphs, but this traditional approach is computationally intensive: even a de novo assembly of small-sized genomes needs millions of reads, making the overlap graph extremely large.
Walking along a Hamiltonian cycle (each vertex once) by following the edges in numerical order allows one to reconstruct the genome by combining alignments between successive reads.
De novo assembly algorithms: Overlap graphs

This method, however, although simple is computationally expensive.
De novo assembly algorithms: Overlap graphs

- A million reads will require a trillion pairwise alignments.
A million reads will require a trillion pairwise alignments.

There is no known efficient algorithm for finding a Hamiltonian cycle.
Figure: The trick is to construct the de Bruijn graph by representing all k-mer prefixes and suffixes as nodes and then drawing edges that represent k-mers having a particular prefix and suffix.
De novo assembly algorithms: de Bruijn graphs

- Most NGS assemblers use de Bruijn graphs.
De novo assembly algorithms: de Bruijn graphs

- Most NGS assemblers use de Bruijn graphs.
- De Bruijn graphs reduce the computational effort by breaking reads into smaller sequences of DNA, called \textit{k-mers}, where the parameter \textit{k} denotes the length in bases of these sequences.
De novo assembly algorithms: de Bruijn graphs

- Most NGS assemblers use de Bruijn graphs.
- De Bruijn graphs reduce the computational effort by breaking reads into smaller sequences of DNA, called \textit{k-mers}, where the parameter \( k \) denotes the length in bases of these sequences.
- The de Bruijn graph captures overlaps of length \( k - 1 \) between these \( k \)-mers and not between the actual reads.
De novo assembly algorithms: de Bruijn graphs

- Most NGS assemblers use de Bruijn graphs.
- De Bruijn graphs reduce the computational effort by breaking reads into smaller sequences of DNA, called \textit{k-mers}, where the parameter \textit{k} denotes the length in bases of these sequences.
- The de Bruijn graph captures overlaps of length \(k - 1\) between these \textit{k}-mers and not between the actual reads.
- By reducing the entire data set down to \textit{k}-mer overlaps the de Bruijn graph reduces redundancy in short-read data sets (same \textit{k}-mers are represented by a unique node in the graph).
De novo assembly algorithms: de Bruijn graphs

- Most NGS assemblers use de Bruijn graphs.
- De Bruijn graphs reduce the computational effort by breaking reads into smaller sequences of DNA, called *k*-mers, where the parameter *k* denotes the length in bases of these sequences.
- The de Bruijn graph captures overlaps of length *k* − 1 between these *k*-mers and not between the actual reads.
- By reducing the entire data set down to *k*-mer overlaps the de Bruijn graph reduces redundancy in short-read data sets (same *k*-mers are represented by a unique node in the graph).
- The most efficient *k*-mer size for a particular assembly is determined by the read length as well as the error rate; *k* has significant influence on the quality of the assembly.
**De novo assembly algorithms: de Bruijn graphs**

- Most NGS assemblers use de Bruijn graphs.
- De Bruijn graphs reduce the computational effort by breaking reads into smaller sequences of DNA, called *k-mers*, where the parameter *k* denotes the length in bases of these sequences.
- The de Bruijn graph captures overlaps of length *k* − 1 between these *k*-mers and not between the actual reads.
- By reducing the entire data set down to *k*-mer overlaps the de Bruijn graph reduces redundancy in short-read data sets (same *k*-mers are represented by a unique node in the graph).
- The most efficient *k*-mer size for a particular assembly is determined by the read length as well as the error rate; *k* has significant influence on the quality of the assembly.
- Another attractive property of de Bruijn graphs is that repeats in the genome can be collapsed in the graph and do not lead to many spurious overlaps.
De novo assembly algorithms: de Bruijn graphs

Figure: Relationship between the quality score $Q$ and the probability $p$ that the corresponding base call is incorrect; using Sanger (red) and Solexa (black) equations.
De novo assembly algorithms: de Bruijn graphs

Genome: ATGGCGTGCAATG

Eulerian cycle
Visit each edge once
Finding an Eulerian cycle (visit each edge once) allows one to reconstruct the genome by forming an alignment in which each successive $k$-mer (from successive edges) is shifted by one position.
De novo assembly algorithms: de Bruijn graphs

Genome: ATGGCGTGCAATG

Eulerian cycle
Visit each edge once
De novo assembly algorithms: de Bruijn graphs

- Hence we avoid the computationally expensive task of finding a Hamiltonian cycle.
De novo assembly algorithms: de Bruijn graphs
As we visit all edges of the de Bruijin graph, which represent all possible $k$-mers we can spell out a candidate genome; for each edge we traverse, we record the first nucleotide of the $k$-mer assigned to that edge.
De novo assembly: a note for Computer Scientists

A simple formulation of the *de novo* assembly problem as an optimization problem phrases the problem as a classical problem of algorithms on strings: the *Shortest Common Superstring* (SCS) problem.

Input: strings $s_1, s_2, \ldots, s_k$, where $s_i \in \Sigma^*$. 
De novo assembly: a note for Computer Scientists

- A simple formulation of the *de novo* assembly problem as an optimization problem phrases the problem as a classical problem of algorithms on strings: the *Shortest Common Superstring* (SCS) problem.

- Input: strings $s_1, s_2, \ldots, s_k$, where $s_i \in \Sigma^*$.

- Output: the shortest string $s$ containing each $s_i$ as a factor.
De novo assembly: a note for Computer Scientists

- Input: strings $s_1, s_2, \ldots, s_k$, where $s_i \in \Sigma^*$.
- Output: the shortest string $s$ containing each $s_i$ as a factor.
- e.g. given $s_1 = \text{abaab}$, $s_2 = \text{baba}$, $s_3 = \text{aabbb}$, and $s_4 = \text{bbab}$, we want to output $s = \text{bbabaabbb}$.
De novo assembly: a note for Computer Scientists

- A simple formulation of the *de novo* assembly problem as an optimization problem phrases the problem as a classical problem of algorithms on strings: the *Shortest Common Superstring* (SCS) problem.
- Input: strings $s_1, s_2, \ldots, s_k$, where $s_i \in \Sigma^*$.
- Output: the shortest string $s$ containing each $s_i$ as a factor.
- e.g. given $s_1 = \text{abaab}$, $s_2 = \text{baba}$, $s_3 = \text{aabbb}$, and $s_4 = \text{bbab}$, we want to output $s = \text{bbabaabbb}$.
- SCS problem is shown to be NP-complete! (via the Traveling Salesman problem)
Mapping assembly: what is it?

ATTAGCATA... ~3GB

Depth 10 * 3GB = 30GB
Mapping assembly: what is it?

Hundreds of millions of short reads (dozens or hundreds of Gigabytes) must be mapped (aligned) against a reference sequence (3Gb for human).
Mapping assembly: what is it?

Hundreds of millions of short reads (dozens or hundreds of Gigabytes) must be mapped (aligned) against a reference sequence (3Gb for human).

Definition

Given a text $t$ of length $n$, where $t \in \Sigma^+$, $\Sigma = \{A, C, G, T\}$, a set $\{p_1, p_2, \ldots, p_r\}$ of patterns, each of length $m < n$, where $p_i \in \Sigma^+$, for all $1 \leq i \leq r$, and an integer $e < m$, find all the factors of $t$, which are at Hamming distance less than, or equal to, $e$ from $p_i$. 
Mapping assembly: what is it?

Hundreds of millions of short reads (dozens or hundreds of Gigabytes) must be mapped (aligned) against a reference sequence (3Gb for human).

**Definition**

Given a text \( t \) of length \( n \), where \( t \in \Sigma^+ \), \( \Sigma = \{A, C, G, T\} \), a set \( \{p_1, p_2, \ldots, p_r\} \) of patterns, each of length \( m < n \), where \( p_i \in \Sigma^+ \), for all \( 1 \leq i \leq r \), and an integer \( e < m \), find all the factors of \( t \), which are at Hamming distance less than, or equal to, \( e \) from \( p_i \).

where \( \Sigma^+ \) denotes the set of all the strings on the alphabet \( \Sigma \) except the empty string \( \varepsilon \).
Mapping assembly: why not BLAST?

- BLAST reports all significant alignments or typically tens of top-scoring alignments.
- In read mapping, we are typically more interested in the best alignment or best few alignments, covering each region of the query sequence.
- For example, suppose a 1000 bp query sequence consists of a 900 bp segment from one chromosome and a 100 bp segment from another chromosome.
- Further, suppose that 400 bp out of the 900 bp segment is a highly repetitive sequence.
- For BLAST, to know this is a chimeric read, we would need to ask it to report all the alignments of the 400 bp repeat, which is costly and wasteful because in general we are not interested in alignments of short repetitive sequences contained in a longer unique sequence.
The most straightforward way of finding all the occurrences of a read, if no gap is allowed, consists in *sliding* the read along the genome sequence and noting the positions where there exists a match.
The most straightforward way of finding all the occurrences of a read, if no gap is allowed, consists in *sliding* the read along the genome sequence and noting the positions where there exists a match.

Unfortunately, although conceptually simple, this algorithm has a huge complexity.
Mapping assembly: algorithms

- The most straightforward way of finding all the occurrences of a read, if no gap is allowed, consists in *sliding* the read along the genome sequence and noting the positions where there exists a match.

- Unfortunately, although conceptually simple, this algorithm has a huge complexity.

- When gaps are allowed, one has to resort to traditional dynamic programming algorithms, such as the Needleman-Wunsch algorithm.
The most straightforward way of finding all the occurrences of a read, if no gap is allowed, consists in *sliding* the read along the genome sequence and noting the positions where there exists a match.

Unfortunately, although conceptually simple, this algorithm has a huge complexity.

When gaps are allowed, one has to resort to traditional dynamic programming algorithms, such as the Needleman-Wunsch algorithm.

Unfortunately, the complexity becomes even larger.
The most straightforward way of finding all the occurrences of a read, if no gap is allowed, consists in *sliding* the read along the genome sequence and noting the positions where there exists a match.

Unfortunately, although conceptually simple, this algorithm has a huge complexity.

When gaps are allowed, one has to resort to traditional dynamic programming algorithms, such as the Needleman-Wunsch algorithm.

Unfortunately, the complexity becomes even larger.

Therefore, to be efficient, all the methods must rely on some sort of pre-processing.
The most straightforward way of finding all the occurrences of a read, if no gap is allowed, consists in sliding the read along the genome sequence and noting the positions where there exists a match.

Unfortunately, although conceptually simple, this algorithm has a huge complexity.

When gaps are allowed, one has to resort to traditional dynamic programming algorithms, such as the Needleman-Wunsch algorithm.

Unfortunately, the complexity becomes even larger.

Therefore, to be efficient, all the methods must rely on some sort of pre-processing.

i.e. index the genome to provide a direct and fast access to its substrings of a given size, using either hashing-based indexes or Burrows-Wheeler-transform-based indexes.
Mapping assembly algorithms: hashing

- Store the positions of the $k$-mers in an array of linked lists, using a value of $k$ significantly less than the read size, say $k = 9$. 
Mapping assembly algorithms: hashing

- Store the positions of the $k$-mers in an array of linked lists, using a value of $k$ significantly less than the read size, say $k = 9$.

- In terms of space, the problem is tractable since there are, at most, $4^9 = 262144$ different 9-mers in the genome.
Mapping assembly algorithms: hashing

- Store the positions of the $k$-mers in an array of linked lists, using a value of $k$ significantly less than the read size, say $k = 9$.
- In terms of space, the problem is tractable since there are, at most, $4^9 = 262144$ different 9-mers in the genome.
- Select a $k$-mer for each read (a good choice is the leftmost part, because the quality is better) and map it to the genome using the hashing procedure—the seed.
Mapping assembly algorithms: hashing

- Store the positions of the $k$-mers in an array of linked lists, using a value of $k$ significantly less than the read size, say $k = 9$.
- In terms of space, the problem is tractable since there are, at most, $4^9 = 262144$ different 9-mers in the genome.
- Select a $k$-mer for each read (a good choice is the leftmost part, because the quality is better) and map it to the genome using the hashing procedure—the seed.
- For each possible hit, the procedure would then try to map the rest (extend) of the read to the genome (possibly allowing errors, in a Needleman-Wunsch-like algorithm).
Mapping assembly algorithms: hashing

- Store the positions of the $k$-mers in an array of linked lists, using a value of $k$ significantly less than the read size, say $k = 9$.

- In terms of space, the problem is tractable since there are, at most, $4^9 = 262144$ different 9-mers in the genome.

- Select a $k$-mer for each read (a good choice is the leftmost part, because the quality is better) and map it to the genome using the hashing procedure—the seed.

- For each possible hit, the procedure would then try to map the rest (extend) of the read to the genome (possibly allowing errors, in a Needleman-Wunsch-like algorithm).

- This two-steps strategy is called seed and extend.
Mapping assembly algorithms: hashing

- Store the positions of the $k$-mers in an array of linked lists, using a value of $k$ significantly less than the read size, say $k = 9$.
- In terms of space, the problem is tractable since there are, at most, $4^9 = 262144$ different 9-mers in the genome.
- Select a $k$-mer for each read (a good choice is the leftmost part, because the quality is better) and map it to the genome using the hashing procedure—the seed.
- For each possible hit, the procedure would then try to map the rest (extend) of the read to the genome (possibly allowing errors, in a Needleman-Wunsch-like algorithm).
- This two-steps strategy is called seed and extend.
- Drawback is that seeds are usually highly repeated in the reference genome: huge linked lists!
A better approach is to divide each read into $q$ equally-long non-overlapping substrings.
A better approach is to divide each read into $q$ equally-long non-overlapping substrings.

Suppose that one allows for $e$ mismatches.
A better approach is to divide each read into $q$ equally-long non-overlapping substrings.

Suppose that one allows for $e$ mismatches.

At least $q - e$ out of the $q$ substrings can be mapped exactly (in the worst case, the $e$ errors are located in $e$ different substrings, thus leaving $q - e$ substrings without error).
A better approach is to divide each read into $q$ equally-long non-overlapping substrings.

Suppose that one allows for $e$ mismatches.

At least $q - e$ out of the $q$ substrings can be mapped exactly (in the worst case, the $e$ errors are located in $e$ different substrings, thus leaving $q - e$ substrings without error).

The above follows immediately from the pigeon-hole principle and is known as the filtering or partitioning into exact matches strategy.
Mapping assembly algorithms: hashing

- A better approach is to divide each read into $q$ equally-long non-overlapping substrings.
- Suppose that one allows for $e$ mismatches.
- At least $q - e$ out of the $q$ substrings can be mapped exactly (in the worst case, the $e$ errors are located in $e$ different substrings, thus leaving $q - e$ substrings without error).
- The above follows immediately from the pigeon-hole principle and is known as the filtering or partitioning into exact matches strategy.
- The $q - e$ substrings that exactly match the genome constitute an anchor.
A better approach is to divide each read into $q$ equally-long non-overlapping substrings.

Suppose that one allows for $e$ mismatches.

At least $q - e$ out of the $q$ substrings can be mapped exactly (in the worst case, the $e$ errors are located in $e$ different substrings, thus leaving $q - e$ substrings without error).

The above follows immediately from the pigeon-hole principle and is known as the filtering or partitioning into exact matches strategy.

The $q - e$ substrings that exactly match the genome constitute an anchor.

There exist $\binom{q}{q-e}$ possible anchor combinations of the $q$ fragments of a read that we have to check and also extend.
Mapping assembly algorithms: hashing

- A better approach is to divide each read into \( q \) equally-long non-overlapping substrings.
- Suppose that one allows for \( e \) mismatches.
- At least \( q - e \) out of the \( q \) substrings can be mapped exactly (in the worst case, the \( e \) errors are located in \( e \) different substrings, thus leaving \( q - e \) substrings without error).
- The above follows immediately from the *pigeon-hole principle* and is known as the *filtering* or *partitioning into exact matches* strategy.
- The \( q - e \) substrings that exactly match the genome constitute an *anchor*.
- There exist \( \binom{q}{q-e} \) possible anchor combinations of the \( q \) fragments of a read that we have to *check* and also *extend*.
- In practice, for the seed part, we use \( q = 4 \) and \( e = 2 \): \( \binom{4}{4-2} = 6 \) combinations.
Mapping assembly algorithms: BWT

Burrows-Wheeler Transform (BWT) (Burrows and Wheeler, 1994) is an algorithm used in data compression applications such as bzip2.
Mapping assembly algorithms: BWT

- Burrows-Wheeler Transform (BWT) (Burrows and Wheeler, 1994) is an algorithm used in data compression applications such as bzip2.
- It can be applied to create a permanent index of the reference sequence, which may be re-used across mapping runs.
Mapping assembly algorithms: BWT

- Burrows-Wheeler Transform (BWT) (Burrows and Wheeler, 1994) is an algorithm used in data compression applications such as bzip2.
- It can be applied to create a permanent index of the reference sequence, which may be re-used across mapping runs.
  - Consider the $n \times n$ matrix in which each row contains a different cyclic rotation of the original text of length $n$. 
Mapping assembly algorithms: BWT

- Burrows-Wheeler Transform (BWT) (Burrows and Wheeler, 1994) is an algorithm used in data compression applications such as bzip2.
- It can be applied to create a permanent index of the reference sequence, which may be re-used across mapping runs.
  - Consider the $n \times n$ matrix in which each row contains a different cyclic rotation of the original text of length $n$.
  - Sort the rows lexicographically.
Mapping assembly algorithms: BWT

- Burrows-Wheeler Transform (BWT) (Burrows and Wheeler, 1994) is an algorithm used in data compression applications such as bzip2.
- It can be applied to create a permanent index of the reference sequence, which may be re-used across mapping runs.
  - Consider the $n \times n$ matrix in which each row contains a different cyclic rotation of the original text of length $n$.
  - Sort the rows lexicographically.
  - BWT is the rightmost column in the sorted matrix.
Mapping assembly algorithms: BWT

- Burrows-Wheeler Transform (BWT) (Burrows and Wheeler, 1994) is an algorithm used in data compression applications such as bzip2.
- It can be applied to create a permanent index of the reference sequence, which may be re-used across mapping runs.
  - Consider the $n \times n$ matrix in which each row contains a different cyclic rotation of the original text of length $n$.
  - Sort the rows lexicographically.
  - BWT is the rightmost column in the sorted matrix.
- If the text has several repeating substrings, then the BWT will have several places where a single character is repeated; e.g. $\text{BWT(mississippi)} = \text{pssmipissii}$. 
Mapping assembly algorithms: BWT

- Burrows-Wheeler Transform (BWT) (Burrows and Wheeler, 1994) is an algorithm used in data compression applications such as bzip2.
- It can be applied to create a permanent index of the reference sequence, which may be re-used across mapping runs.
  - Consider the $n \times n$ matrix in which each row contains a different cyclic rotation of the original text of length $n$.
  - Sort the rows lexicographically.
  - BWT is the rightmost column in the sorted matrix.
- If the text has several repeating substrings, then the BWT will have several places where a single character is repeated; e.g. $\text{BWT(mississippi)} = \text{pssmipissii}$.
- The remarkable thing about the BWT is that it is reversible—allowing the original text to be re-generated only from the last column!
# Mapping assembly algorithms: BWT

<table>
<thead>
<tr>
<th>mississippi</th>
<th>imississipp</th>
<th>pimississip</th>
<th>ppimississi</th>
<th>ippimississ</th>
<th>sippimississ</th>
<th>ssippimississi</th>
<th>issippimississ</th>
<th>sissippimississ</th>
<th>ssissippimississ</th>
<th>ississippimississ</th>
<th>sississippimississ</th>
<th>ssississippimississ</th>
<th>issississippimississ</th>
</tr>
</thead>
</table>

**Table:** $n \times n$ matrix of the cyclic rotations of *mississippi*
Mapping assembly algorithms: BWT

<table>
<thead>
<tr>
<th>Prefix of $n - 1$ letters</th>
<th>$n$th letter (BWT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>imississipp</td>
<td>p</td>
</tr>
<tr>
<td>ippimississ</td>
<td>s</td>
</tr>
<tr>
<td>issippimis</td>
<td>s</td>
</tr>
<tr>
<td>ississippi</td>
<td>m</td>
</tr>
<tr>
<td>mississippi</td>
<td>i</td>
</tr>
<tr>
<td>pimississi</td>
<td>p</td>
</tr>
<tr>
<td>ppimississi</td>
<td>i</td>
</tr>
<tr>
<td>sippissi</td>
<td>s</td>
</tr>
<tr>
<td>sissippimi</td>
<td>s</td>
</tr>
<tr>
<td>ssippimiss</td>
<td>i</td>
</tr>
<tr>
<td>ssisissippim</td>
<td>i</td>
</tr>
</tbody>
</table>

Table: $n \times n$ lexicographically sorted matrix of the cyclic rotations of mississippi
Mapping assembly algorithms: BWT

- There exists a direct relationship between the BWT and the suffix array—an efficient indexing data structure from which we may obtain directly the BWT.
Mapping assembly algorithms: BWT

- There exists a direct relationship between the BWT and the suffix array—an efficient indexing data structure from which we may obtain directly the BWT.
- The amount of storage that we need to store the BWT, however, is *significantly smaller* than that suffix array.
There exists a direct relationship between the BWT and the suffix array—an efficient indexing data structure from which we may obtain directly the BWT.

The amount of storage that we need to store the BWT, however, is significantly smaller than that suffix array.

An increasing number of algorithms is developed to search these compressed full-text indexes for permitting fast substring queries; the most well-known is the FM-index (Ferragina and Manzini, 2000).
Mapping assembly algorithms: BWT

- There exists a direct relationship between the BWT and the suffix array—an efficient indexing data structure from which we may obtain directly the BWT.
- The amount of storage that we need to store the BWT, however, is significantly smaller than that suffix array.
- An increasing number of algorithms is developed to search these compressed full-text indexes for permitting fast substring queries; the most well-known is the FM-index (Ferragina and Manzini, 2000).
- It can be used to efficiently find the number of occurrences of a pattern within the compressed text, as well as to locate the position of each occurrence.
There exists a direct relationship between the BWT and the suffix array—an efficient indexing data structure from which we may obtain directly the BWT.

The amount of storage that we need to store the BWT, however, is significantly smaller than that suffix array.

An increasing number of algorithms is developed to search these compressed full-text indexes for permitting fast substring queries; the most well-known is the FM-index (Ferragina and Manzini, 2000).

It can be used to efficiently find the number of occurrences of a pattern within the compressed text, as well as to locate the position of each occurrence.

Both the query time and storage space requirements are sublinear with respect to the size of the input data.
Introduction

BLAST

Genome assembly

Conclusion

Mapping assembly

Mapping assembly algorithms: BWT

- There exists a direct relationship between the BWT and the suffix array—an efficient indexing data structure from which we may obtain directly the BWT.
- The amount of storage that we need to store the BWT, however, is *significantly smaller* than that suffix array.
- An increasing number of algorithms is developed to search these compressed full-text indexes for permitting fast *substring queries*; the most well-known is the *FM-index* (Ferragina and Manzini, 2000).
- It can be used to efficiently *find the number of occurrences* of a pattern within the compressed text, as well as to *locate the position* of each occurrence.
- Both the query time and storage space requirements are *sublinear* with respect to the size of the input data.
- Most recent mapping tools are based on such BWT indexes.
Mapping assembly algorithms: some experiments

**Table:** Mapping 25,000,000 64 bp-long simulated reads to the human chromosome 6 (166,880,988 bp)

<table>
<thead>
<tr>
<th>Programme</th>
<th>Total time</th>
<th>Reads aligned</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Indexing</td>
<td>Mapping</td>
</tr>
<tr>
<td>SOAP2</td>
<td>5m10s</td>
<td>28m25s</td>
</tr>
<tr>
<td>REAL -q 0</td>
<td>0m00s</td>
<td>26m43s</td>
</tr>
<tr>
<td>Bowtie</td>
<td>7m35s</td>
<td>49m11s</td>
</tr>
<tr>
<td>REAL -q 1</td>
<td>0m00s</td>
<td>31m54s</td>
</tr>
</tbody>
</table>

All programmes were run with 48 bp-long seed, with at most two mismatches in the seed, and reported best hits only.
Mapping assembly algorithms: some experiments

**Table:** Mapping 24,543,488 70 bp-long simulated reads to the Drosophila melanogaster chromosome 3L (24,543,557 bp)

<table>
<thead>
<tr>
<th>Programme</th>
<th>Total time</th>
<th>Reads aligned</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Indexing</td>
<td>Mapping</td>
<td></td>
</tr>
<tr>
<td>SOAP2</td>
<td>0m45s</td>
<td>16m02s</td>
<td>21,126,303</td>
</tr>
<tr>
<td>REAL -q 0</td>
<td>0m00s</td>
<td>10m44s</td>
<td>21,134,692</td>
</tr>
<tr>
<td>Bowtie</td>
<td>0m59s</td>
<td>40m28s</td>
<td>18,920,716</td>
</tr>
<tr>
<td>REAL -q 1</td>
<td>0m00s</td>
<td>15m42s</td>
<td>21,134,699</td>
</tr>
</tbody>
</table>

All programmes were run with 48 bp-long seed, with at most two mismatches in the seed, and reported the best hits only.
## Mapping assembly algorithms: some experiments

<table>
<thead>
<tr>
<th>Programme</th>
<th>Total time</th>
<th>Reads aligned</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Indexing</td>
<td>Mapping</td>
</tr>
<tr>
<td>SOAP2</td>
<td>1h58m07s</td>
<td>1h52m21s</td>
</tr>
<tr>
<td>REAL -q 0</td>
<td>0m00s</td>
<td>4h08m47s</td>
</tr>
<tr>
<td>Bowtie</td>
<td>3h29m59s</td>
<td>1h56m41s</td>
</tr>
<tr>
<td>REAL -q 1</td>
<td>0m00s</td>
<td>4h20m37s</td>
</tr>
</tbody>
</table>

All programmes were run with 48 bp-long seed, with at most two mismatches in the seed, and reported the best hits only.
Contents

1 Introduction
2 Basic definitions
3 Alignment algorithms on strings
4 Conclusion
Overview

- **BLAST**: a set of programs for the comparison of biological sequences.
Overview

- BLAST: a set of programs for the comparison of biological sequences.
- Recent technological advances have dramatically improved next-generation sequencing throughput and quality.
Overview

- **BLAST**: a set of programs for the comparison of biological sequences.
- Recent technological advances have dramatically improved next-generation sequencing throughput and quality.
- In parallel with the technological improvements that have increased the throughput of the next-generation short-read sequencers, many algorithmic advances have been made.
Overview

- **BLAST**: a set of programs for the comparison of biological sequences.
- Recent technological advances have dramatically improved next-generation sequencing throughput and quality.
- In parallel with the technological improvements that have increased the throughput of the next-generation short-read sequencers, many algorithmic advances have been made.
- Genome assembly: taking a huge number of DNA sequences and putting them back together to create a representation of the genome from which the DNA originated.
Overview

- BLAST: a set of programs for the comparison of biological sequences.
- Recent technological advances have dramatically improved next-generation sequencing throughput and quality.
- In parallel with the technological improvements that have increased the throughput of the next-generation short-read sequencers, many algorithmic advances have been made.
- Genome assembly: taking a huge number of DNA sequences and putting them back together to create a representation of the genome from which the DNA originated.
- De novo assembly: assembling short reads to create full-length—sometimes novel—sequences.
Overview

- **BLAST**: a set of programs for the comparison of biological sequences.
- Recent technological advances have dramatically improved next-generation sequencing throughput and quality.
- In parallel with the technological improvements that have increased the throughput of the next-generation short-read sequencers, many algorithmic advances have been made.
- Genome assembly: taking a huge number of DNA sequences and putting them back together to create a representation of the genome from which the DNA originated.
- *De novo* assembly: assembling short reads to create full-length—sometimes novel—sequences.
- Mapping assembly: assembling reads by aligning them against an existing reference sequence—building a sequence that is similar but not necessarily identical to the reference.
S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman.
Basic Local Alignment Search Tool.  

M. Burrows and D. J. Wheeler. 
A block-sorting lossless data compression algorithm.  

J. C. Dohm, C. Lottaz, T. Borodina, and H. Himmelbauer. 
SHARCGS, a fast and highly accurate short-read assembly algorithm for *de novo* genomic sequencing.  

P. Ferragina and G. Manzini. 
Opportunistic data structures with applications. 

Whole-genome random sequencing and assembly of *Haemophilus influenzae*.


REAL: an efficient REad ALigner for next generation sequencing reads.

De novo bacterial genome sequencing: millions of very short reads assembled on a desktop computer.


R. Li, H. Zhu, J. Ruan, W. Qian, X. Fang, Z. Shi, Y. Li, S. Li, G. Shan, K. Kristiansen, S. Li, H. Yang, J. Wang, and J. Wang.

De novo assembly of human genomes with massively parallel short read sequencing.


Genome sequencing in microfabricated high-density picolitre reactors.

Assembly algorithms for next-generation sequencing data.

Accurate Multiplex Polony Sequencing of an Evolved Bacterial Genome.

J. R. ten Bosch and W. W. Grody.
Keeping up with the next generation: Massively parallel sequencing in clinical diagnostics.

*Journal of Molecular Diagnostics, 10(6):484–492, 2008.*