BOX

Genome assembly (*or on modeling biological problems in CS*)

Léo Ackermann

N.B. Credits of figures are missing

Before starting

If you want to contact me. By email (leo.ackermann@inria.fr) or by Discord (tag me)

A few review you may find interesting. (Theoretical questions related to bioinformatics)

- *Parameterized Algorithms in Bioinformatics: An Overview.* Bulteau & Weller, 2019.
- *Theoretical analysis of edit distance algorithms. Medvedev, 2023.*
- *Theoretical analysis of sequencing bioinformatics algorithms and beyond. Medvedev, 2023.*
- *Modeling Biological Problems in Computer Science: A Case Study in Genome Assembly. Medvedev, 2018.*
- *Information theory in computational biology: where we stand today. Chanda et al. 2020.*
- *Sketching and sublinear data structures in genomics. Marçais et al. 2019.*
- *When less is more: sketching with minimizers in genomics. Ndiaye et al. 2024.*
- *Advancements in practical k-mer sets: essentials for the curious. Marchet. 2024.*
- *Indexing highly repetitive string collections, part I: repetitiveness measures. Navarro. 2021.*
- *Indexing highly repetitive string collections, part II: compressed indexes. Navarro. 2021.*

*blueprint of this lecture

Today's program

A. Discovering genome assembly

B. Trying to formulate it as a mathematical/CS problem

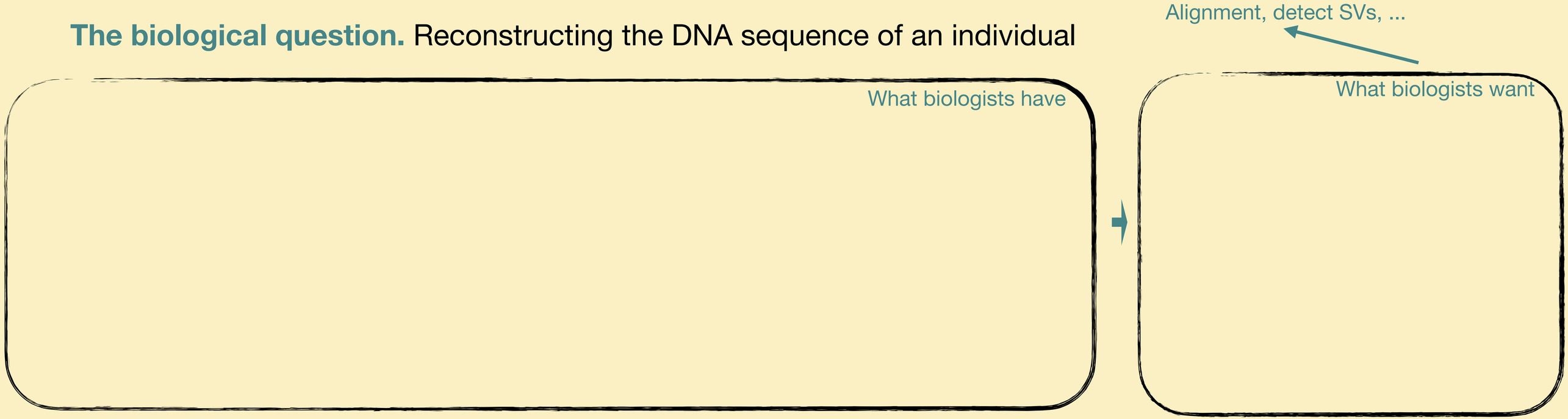
C. Discovering the two main approaches used in practice to solve this problem

Part I

Modeling the biological question

The *De Novo* Sequence Assembly problem

The biological question. Reconstructing the DNA sequence of an individual



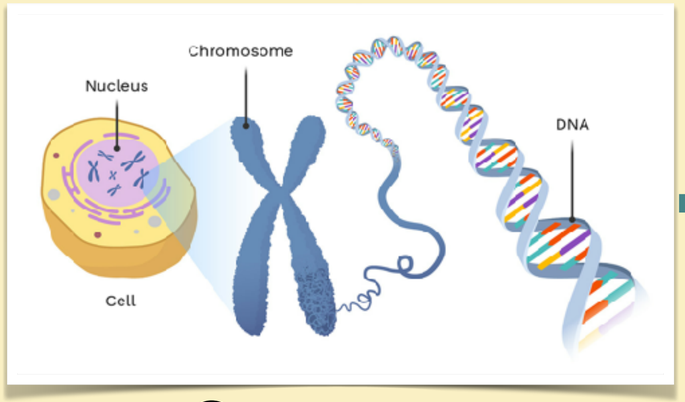
The *De Novo* Sequence Assembly problem

The biological question. Reconstructing the DNA sequence of an individual

Alignment, detect SVs, ...

What biologists have

What biologists want



Genome

Unknown DNA "strings"

DNA Sequencing

Reads

DNA "substrings"

(1M of ±200nt reads for a 5M-nt genome)

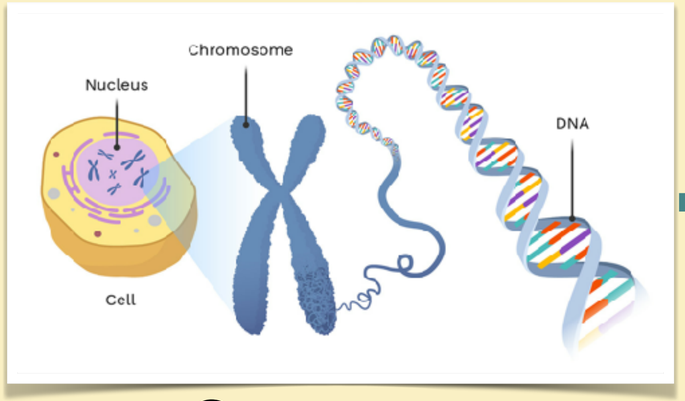
The *De Novo* Sequence Assembly problem

The biological question. Reconstructing the DNA sequence of an individual

Alignment, detect SVs, ...

What biologists have

What biologists want



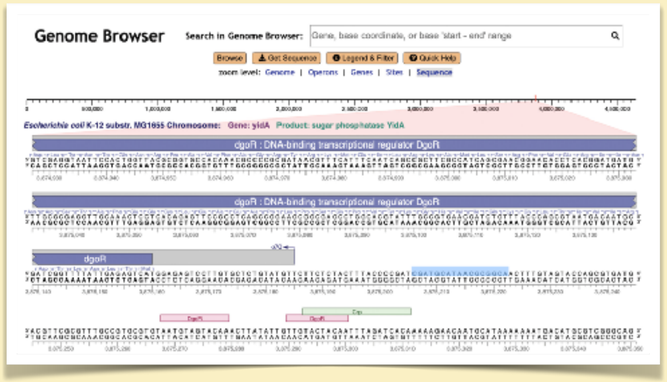
Genome

Unknown DNA "strings"

DNA Sequencing

Reads

DNA "substrings"
(1M of ±200nt reads for a 5M-nt genome)



Assembly

Computational representation of genome

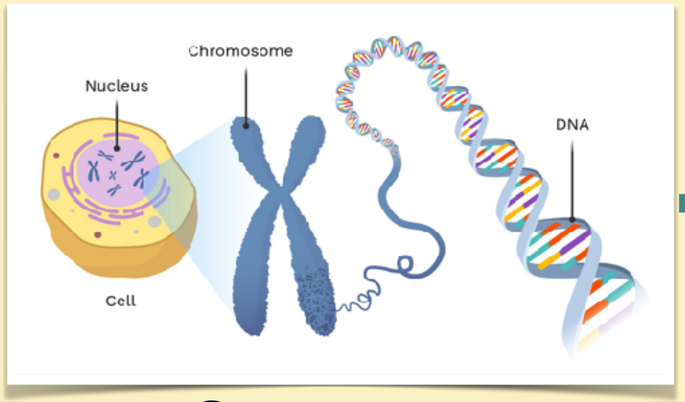
The *De Novo* Sequence Assembly problem

The biological question. Reconstructing the DNA sequence of an individual

Alignment, detect SVs, ...

What biologists have

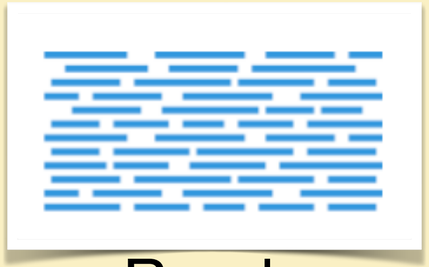
What biologists want



Genome

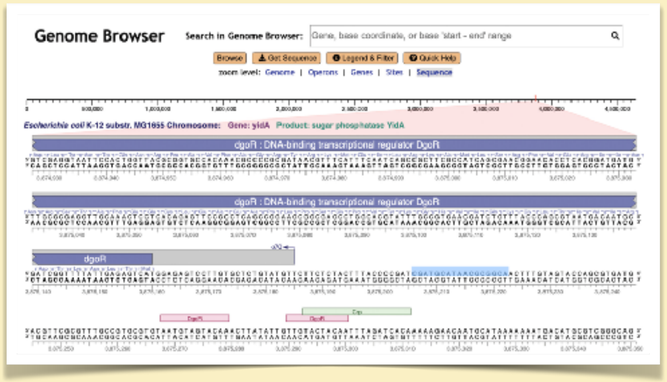
Unknown DNA "strings"

DNA Sequencing



Reads

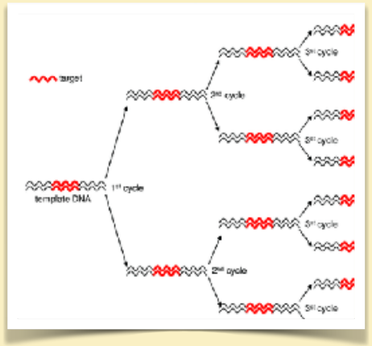
DNA "substrings"
(1M of ±200nt reads for a 5M-nt genome)



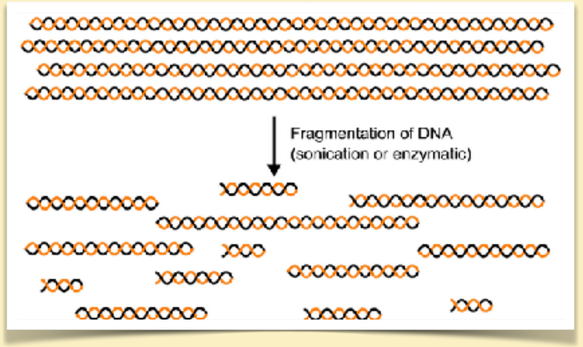
Assembly

Computational representation of genome

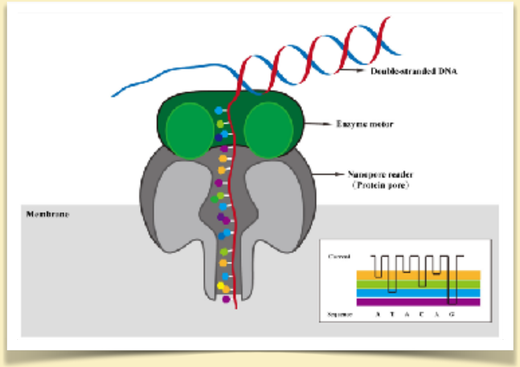
A (quick) zoom on DNA Sequencing.



DNA Amplification



DNA Fragmentation



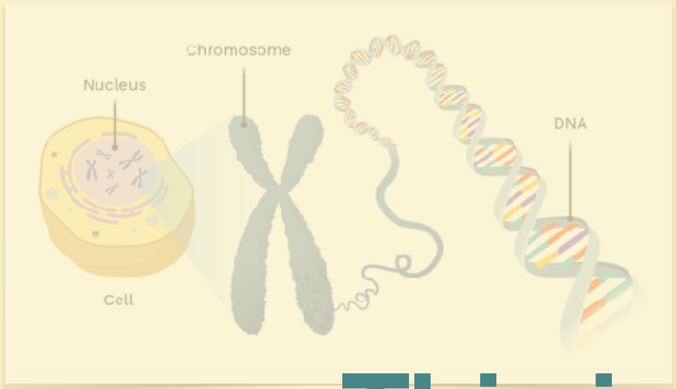
Signal acquisition

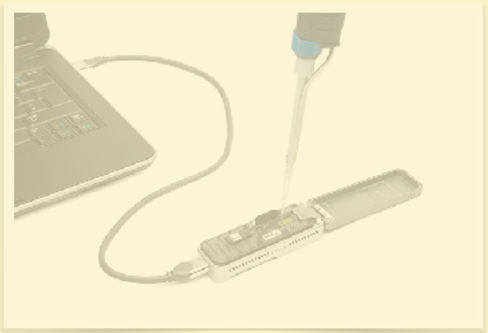
(+ base calling)

The *De Novo* Sequence Assembly problem

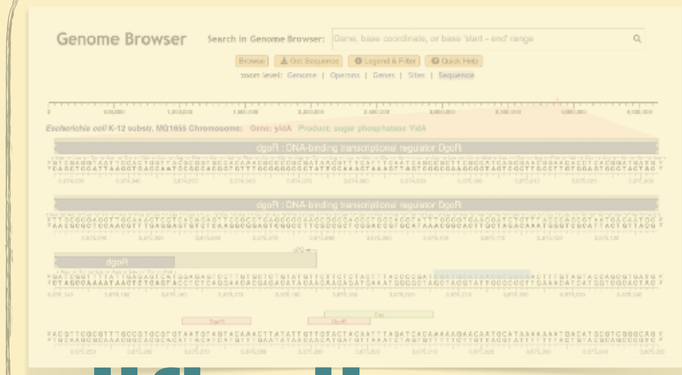
The biological question. Reconstructing the DNA sequence of an individual

What biologists have





What biologists want



Genome

Unknown DNA "strings"

DNA Sequencing

Reads

DNA "substrings"

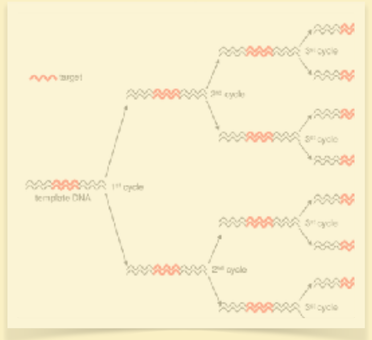
Millions of ~200nt reads for a ~5-nt genome

Assembly

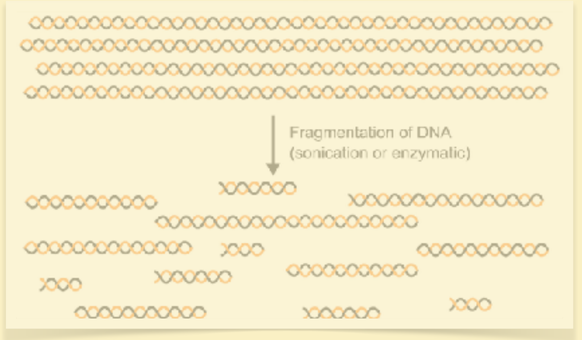
Computational representation of genome

This is already extremely simplified!

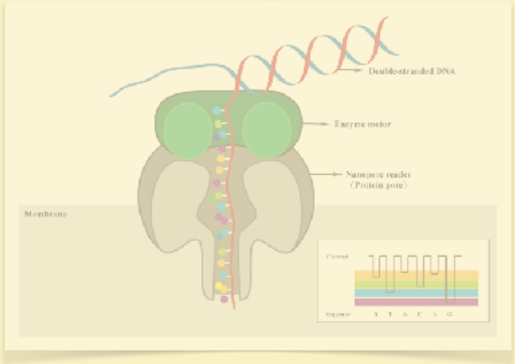
A (quick) zoom on DNA Sequencing.



DNA Amplification



DNA Fragmentation



Signal acquisition

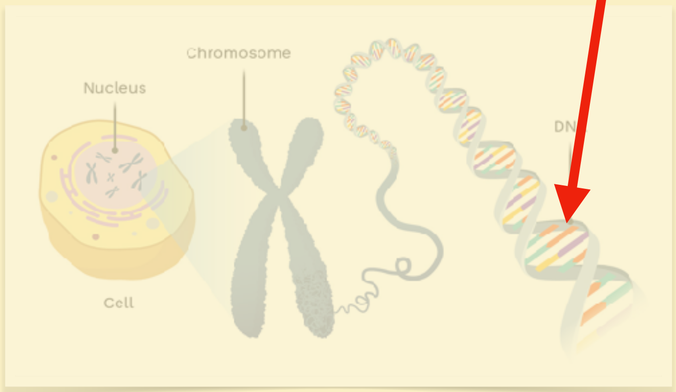
(+ base calling)

The *De Novo* Sequence Assembly problem

[1] Double-strands vs single-strand

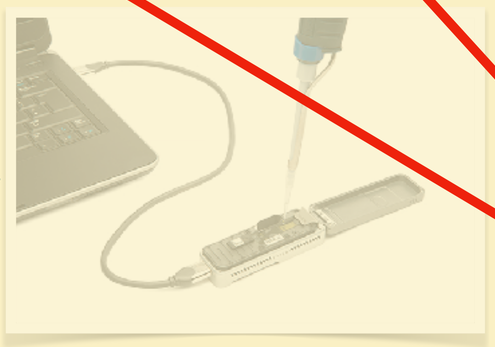
The biological question: Reconstructing the DNA sequence of an individual

What biologists have



Genome

Unknown DNA "strings"



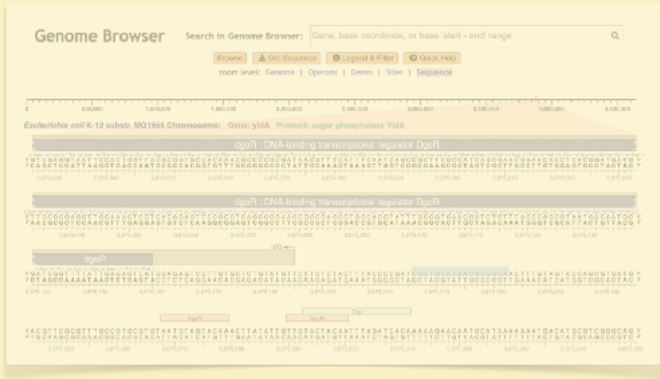
DNA Sequencing

Reads

DNA "substrings"

(1M of ±200nt reads for a 5M-nt genome)

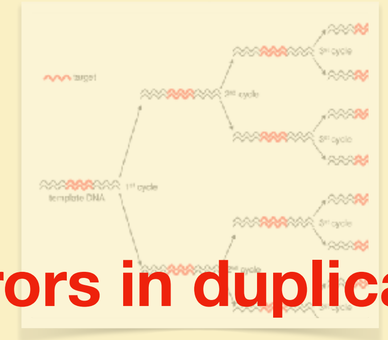
What biologists want



Assembly

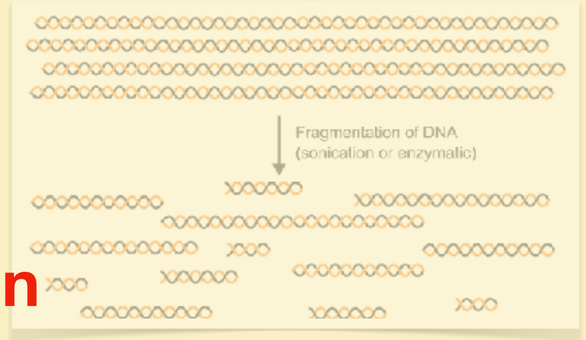
Computational representation of genome

A (quick) zoom on DNA Sequencing.

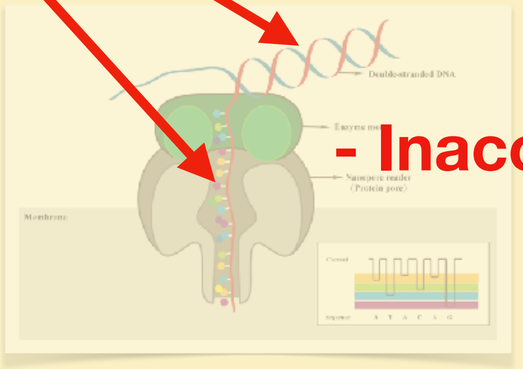


DNA Amplification

[2] Errors in duplication



DNA Fragmentation



- Inaccuracies

Signal acquisition

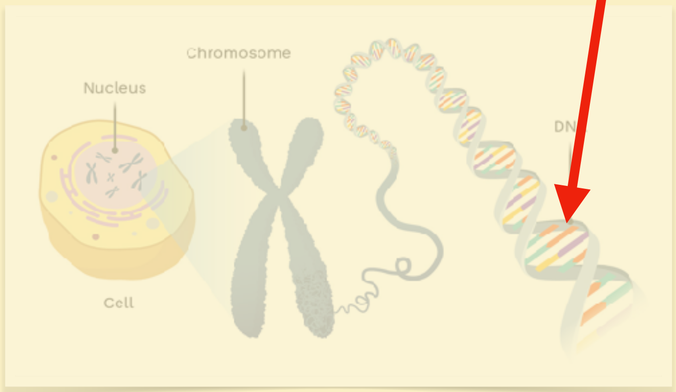
(+ base calling)

The *De Novo* Sequence Assembly problem

[1] Double-strands vs single-strand

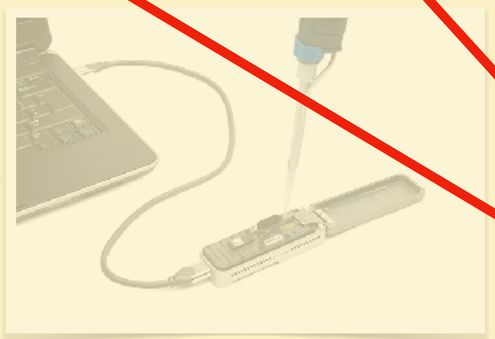
The biological question: Reconstructing the DNA sequence of an individual

What biologists have

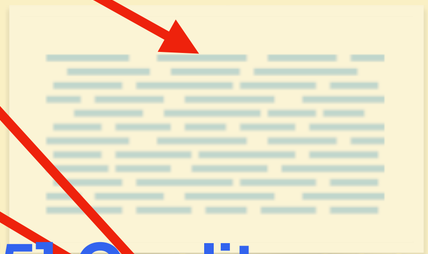


Genome

Unknown DNA "strings"



DNA Sequencing



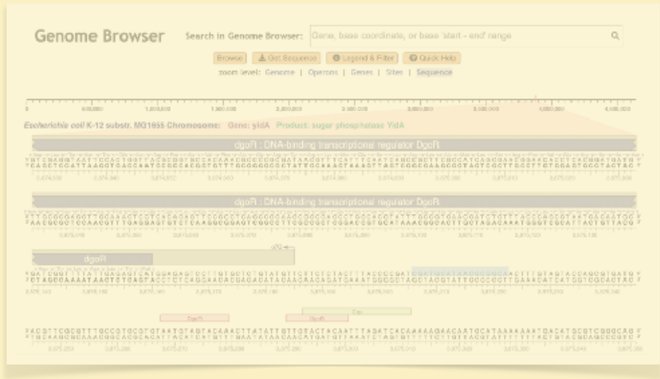
Reads

DNA "substrings"

(1M of ±200nt reads for a 5M-nt genome)

[5] Quality score

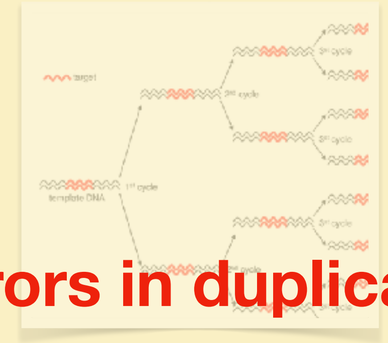
What biologists want



Assembly

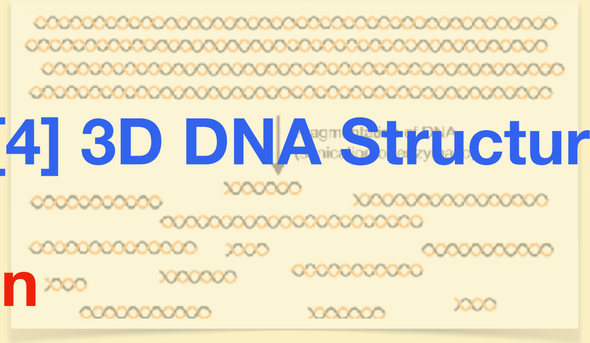
Computational representation of genome

A (quick) zoom on DNA Sequencing.



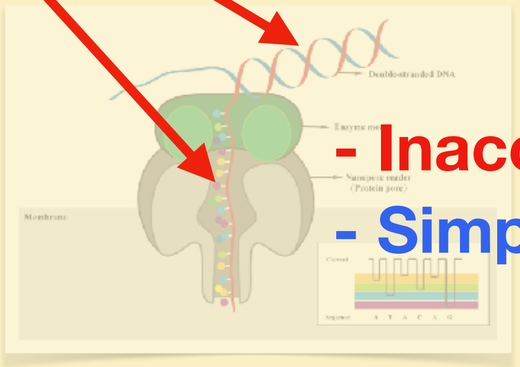
DNA Amplification

[2] Errors in duplication



DNA Fragmentation

[4] 3D DNA Structure



Signal acquisition

(+ base calling)

- Inaccuracies

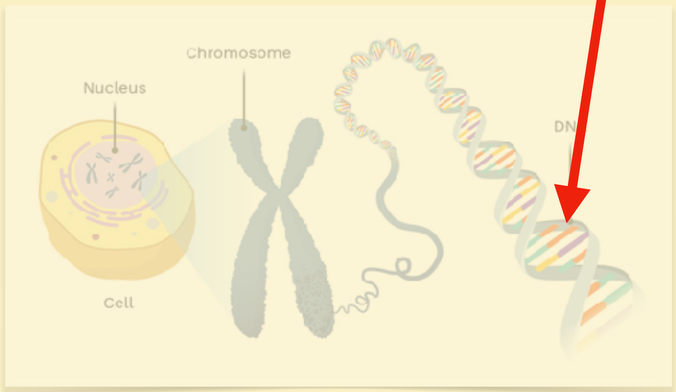
- Simplifications

The *De Novo* Sequence Assembly problem

[1] Double-strands vs single-strand

The biological question: Reconstructing the DNA sequence of an individual

What biologists have

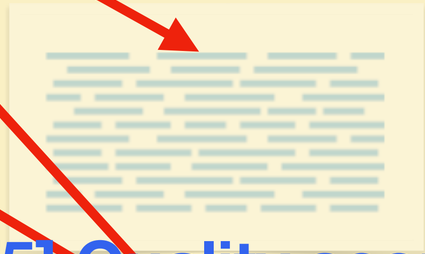


Genome

Unknown DNA "strings"

[3] Human errors

DNA Sequencing



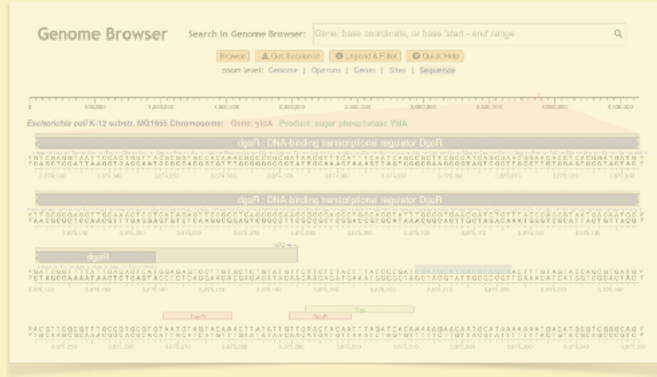
Reads

DNA "substrings"

(1M of ±200nt reads for a 5M-nt genome)

[5] Quality score

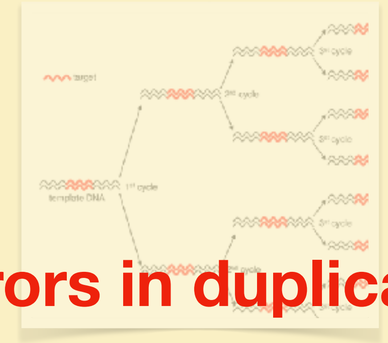
What biologists want



Assembly

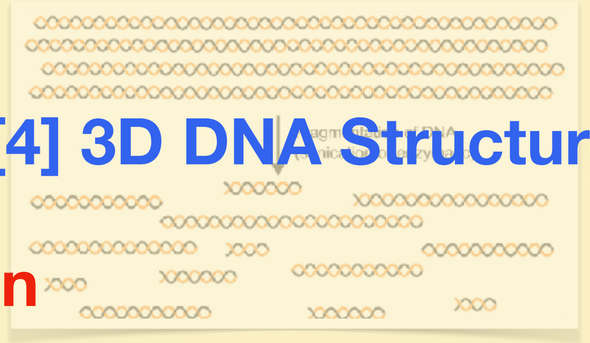
Computational representation of genome

A (quick) zoom on DNA Sequencing.



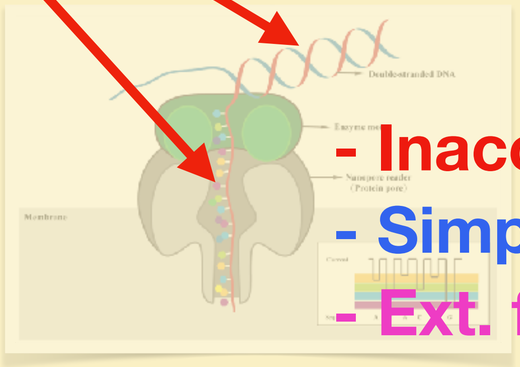
[2] Errors in duplication

DNA Amplification



[4] 3D DNA Structure

DNA Fragmentation



Signal acquisition

(+ base calling)

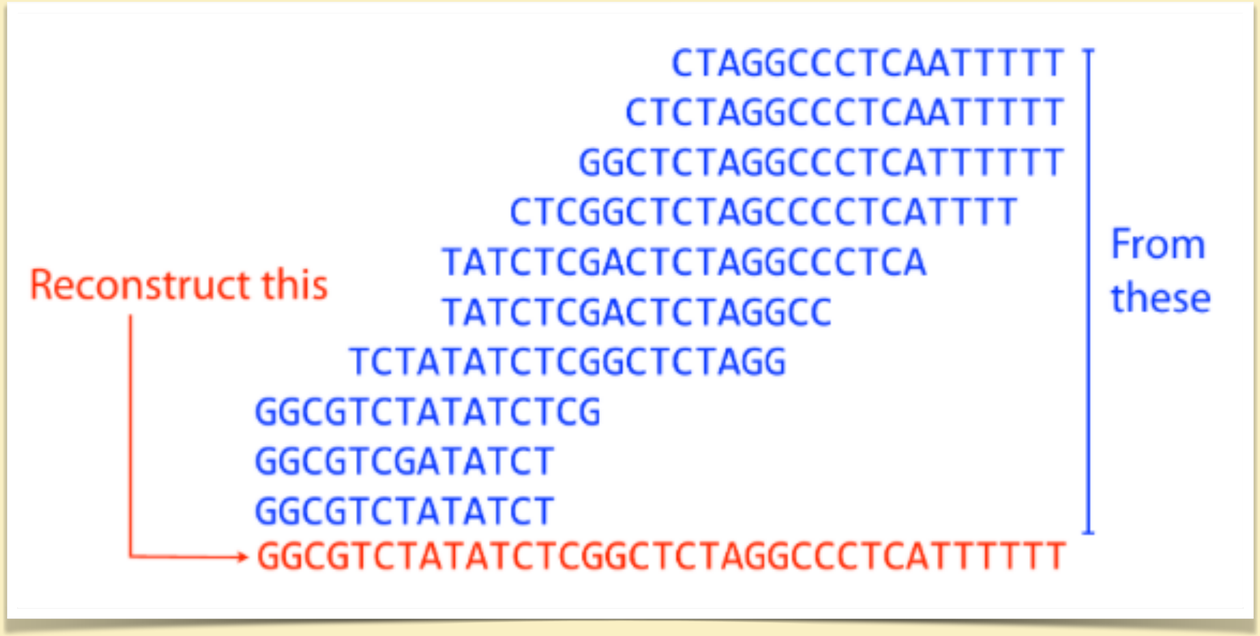
- Inaccuracies

- Simplifications

- Ext. factors

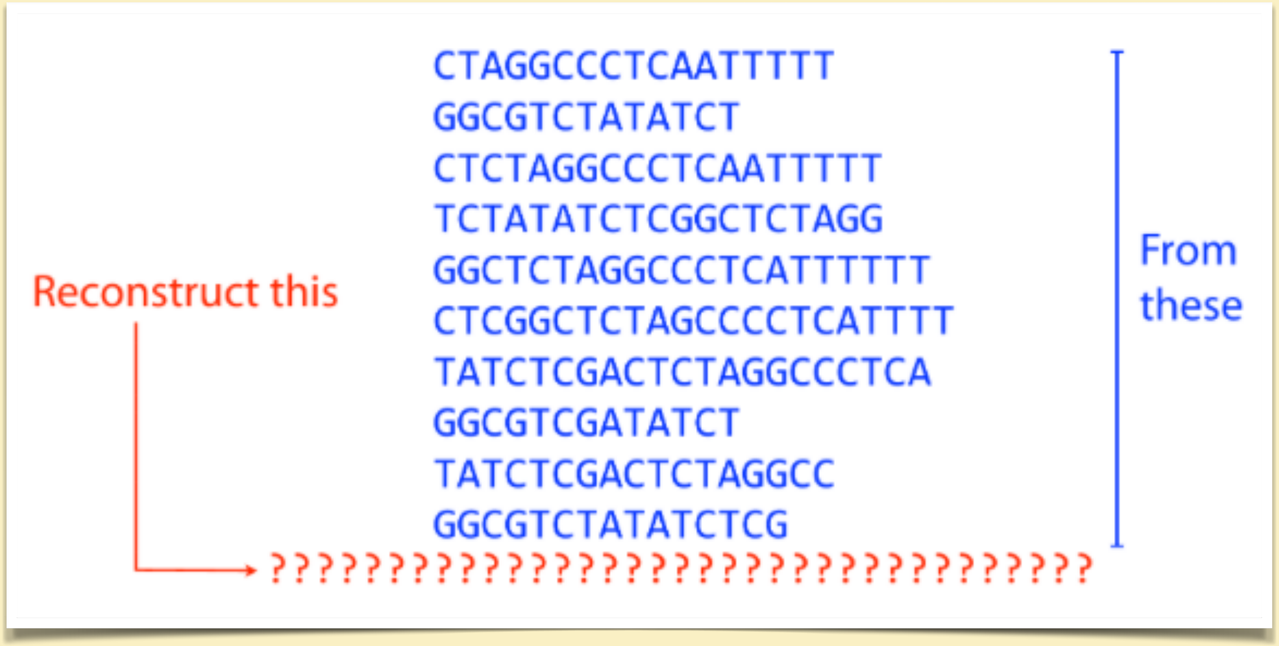
The *De Novo* Sequence Assembly problem

A computer science perspective (draft).



The *De Novo* Sequence Assembly problem

A computer science perspective (draft).



A well-formulated problem

Toward a well-defined problem

Input: A collection of strings $\mathcal{S} = \{s_1, \cdots, s_n\}$ generated by a sequencing experiment on some genome

Output: the genome that generated $\mathcal{S}$

What do you think of it?

Toward a well-defined problem

Input: A collection of strings $\mathcal{S} = \{s_1, \cdots, s_n\}$ generated by a sequencing experiment on some genome

Output: the genome that generated $\mathcal{S}$

GOOD. The input and output of the problem are precisely defined

PROBLEM. The problem is not self-contained ("sequencing experiment", "genome")

Toward a well-defined problem

Input: A collection of strings $\mathcal{S} = \{s_1, \cdots, s_n\}$ generated by a sequencing experiment on some genome

Output: the genome that generated $\mathcal{S}$

GOOD. The input and output of the problem are precisely defined

PROBLEM. The problem is not self-contained ("sequencing experiment", "genome")

Input: a collection of strings $\mathcal{S} = \{s_1, \cdots, s_n\}$, all of them being substring of some unknown string G

Output: the unknown string G

What do you think of it?

Toward a well-defined problem

Input: A collection of strings $\mathcal{S} = \{s_1, \cdots, s_n\}$ generated by a sequencing experiment on some genome

Output: the genome that generated $\mathcal{S}$

GOOD. The input and output of the problem are precisely defined

PROBLEM. The problem is not self-contained ("sequencing experiment", "genome")

Input: a collection of strings $\mathcal{S} = \{s_1, \cdots, s_n\}$, all of them being substring of some unknown string G

Output: the unknown string G

PROBLEM. No way to check whether the output of a program that solves it is correct

Toward a well-defined problem

Input: A collection of strings $\mathcal{S} = \{s_1, \cdots, s_n\}$ generated by a sequencing experiment on some genome
Output: the genome that generated $\mathcal{S}$

GOOD. The input and output of the problem are precisely defined
PROBLEM. The problem is not self-contained ("sequencing experiment", "genome")

Input: a collection of strings $\mathcal{S} = \{s_1, \cdots, s_n\}$, all of them being substring of some unknown string G
Output: the unknown string G

PROBLEM. No way to check whether the output of a program that solves it is correct

Definition (Genome assembly, v1).

Input: a collection of strings $\mathcal{S} = \{s_1, \cdots, s_n\}$
Output: a string G that is a common superstring of $\mathcal{S}$

About simplifications

Simple formulations of problems are likely easier to solve/study. Three approaches:

About simplifications

Simple formulations of problems are likely easier to solve/study. Three approaches:

[A] Use simplified formulations of problems, and hope that it does not affect the accuracy on real data

About simplifications

Simple formulations of problems are likely easier to solve/study. Three approaches:

[A] Use simplified formulations of problems, and hope that it does not affect the accuracy on real data

[B] Use more complex formulations of problems, and hope that we can still come up with an efficient algorithm

About simplifications

Simple formulations of problems are likely easier to solve/study. Three approaches:

[A] Use simplified formulations of problems, and hope that it does not affect the accuracy on real data

[B] Use more complex formulations of problems, and hope that we can still come up with an efficient algorithm

[C] Modularize the problem formulation (eg. Step1: ensure a simplifying assumption, Step2: solve it with the simplifying assumption)

About simplifications

Simple formulations of problems are likely easier to solve/study. Three approaches:

[A] Use simplified formulations of problems, and hope that it does not affect the accuracy on real data

[B] Use more complex formulations of problems, and hope that we can still come up with an efficient algorithm

[C] Modularize the problem formulation (eg. Step1: ensure a simplifying assumption, Step2: solve it with the simplifying assumption)

Motto. **The ultimate test of a (practical) algorithm is how is performs on real data!**

A useful (?) problem

Optimization criterion

Definition (Genome assembly, v1).

Input: a collection of strings $\mathcal{S} = \{s_1, \cdots, s_n\}$

Output: a string G that is a common superstring of $\mathcal{S}$

Example. Consider $\mathcal{S} = \{ACG, CGT\}$. What is G?

Optimization criterion

Definition (Genome assembly, v1).

Input: a collection of strings $\mathcal{S} = \{s_1, \cdots, s_n\}$

Output: a string G that is a common superstring of $\mathcal{S}$

Example. Consider $\mathcal{S} = \{ACG, CGT\}$. What is G?

Many possible answers: ACGT, ACGCGT, ACGTTTTTTTTCGT, CGTACACACG, ...

Which one should we pick?

Optimization criterion

Definition (Genome assembly, v1).

Input: a collection of strings $\mathcal{S} = \{s_1, \cdots, s_n\}$

Output: a string G that is a common superstring of $\mathcal{S}$

Example. Consider $\mathcal{S} = \{ACG, CGT\}$. What is G?

Many possible answers: ACGT, ACGCGT, ACGTTTTTTTTCGT, CGTACACACG, ...

"the simplest explanation is likely the correct one"

One possible choice. Take one shortest such string, guided by the parsimony principle

Optimization criterion

Definition (Genome assembly, v1).

Input: a collection of strings $\mathcal{S} = \{s_1, \cdots, s_n\}$

Output: a string G that is a common superstring of $\mathcal{S}$

Example. Consider $\mathcal{S} = \{ACG, CGT\}$. What is G?

Many possible answers: ACGT, ACGCGT, ACGTTTTTTTTCGT, CGTACACACG, ...

"the simplest explanation is likely the correct one"

One possible choice. Take one shortest such string, guided by the parsimony principle

Definition (Genome assembly, v2).

Input: a collection of strings $\mathcal{S} = \{s_1, \cdots, s_n\}$

Output: a string G that is a shortest common superstring of $\mathcal{S}$

Optimization criterion

Definition (Genome assembly, v1).

Input: a collection of strings $\mathcal{S} = \{s_1, \cdots, s_n\}$

Output: a string G that is a common superstring of $\mathcal{S}$

Example. Consider $\mathcal{S} = \{ACG, CGT\}$. What is G?

Many possible answers: ACGT, ACGCGT, ACGTTTTTTTTCGT, CGTACACACG, ...

"the simplest explanation is likely the correct one"

One possible choice. Take one shortest such string, guided by the parsimony principle

Definition (Genome assembly, v2).

Input: a collection of strings $\mathcal{S} = \{s_1, \cdots, s_n\}$

Output: a string G that is a shortest common superstring of $\mathcal{S}$

Limitation. This doesn't guarantee the unicity of the solution (but maybe not a problem on real data).

Computational complexity

Lemma. The "Genome Assembly (v2)" is NP-complete (by reduction to Hamiltonian path)

Computational complexity

Lemma. The "Genome Assembly (v2)" is NP-complete (by reduction to Hamiltonian path)

This is not (necessarily) the end of the world!

Why?

Computational complexity

Lemma. The "Genome Assembly (v2)" is NP-complete (by reduction to Hamiltonian path)

This is not (necessarily) the end of the world!

NP-complete: it is unlikely that there exists an algorithm that solves any instance in polytime

Computational complexity

Lemma. The "Genome Assembly (v2)" is NP-complete (by reduction to Hamiltonian path)

This is not (necessarily) the end of the world!

NP-complete: it is unlikely that there exists an algorithm that solves any instance in polytime

1. It is possible that "real world instances" are not worst-case inputs for the problem (eg. Horn-clauses for SAT)
2. It is possible that a heuristics algorithm performs well on "real world instances" (eg. SAT-solvers)
3. It is possible that the "real world instances" are small enough to run an exponential time algorithms

Computational complexity

Lemma. The "Genome Assembly (v2)" is NP-complete (by reduction to Hamiltonian path)

This is not (necessarily) the end of the world!

NP-complete: it is unlikely that there exists an algorithm that solves any instance in polytime

1. It is possible that "real world instances" are not worst-case inputs for the problem (eg. Horn-clauses for SAT)
2. It is possible that a heuristics algorithm performs well on "real world instances" (eg. SAT-solvers)
3. It is possible that the "real world instances" are small enough to run an exponential time algorithms

Still, ideally we would like a tractable formulation of the problem.

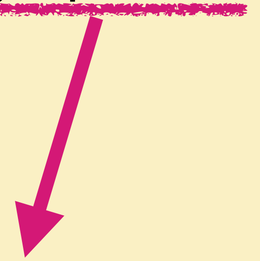
Computational complexity (critics)

Computational complexity (critics)

Idea. Asymptotical worst-case time complexity in the RAM model is not a perfect compass

Computational complexity (critics)

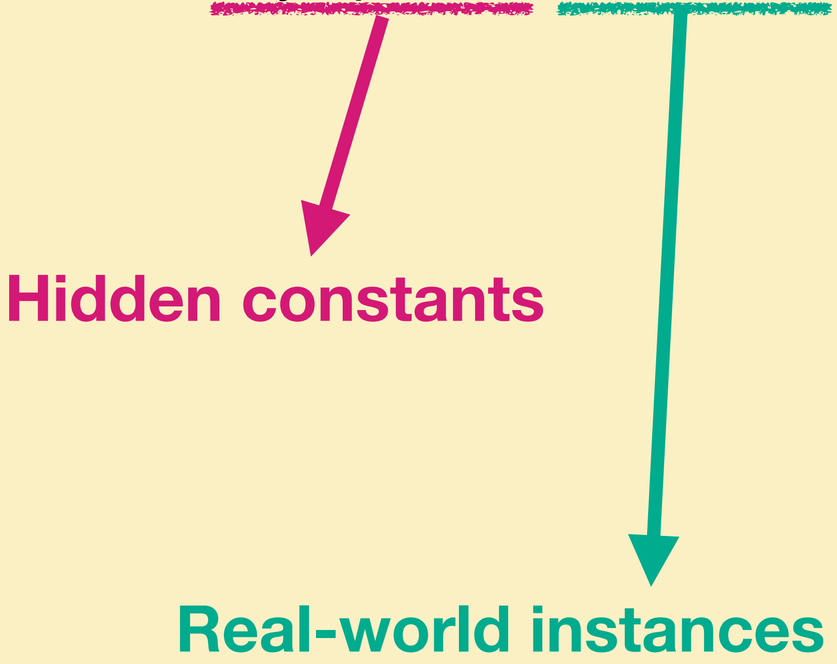
Idea. Asymptotical worst-case time complexity in the RAM model is not a perfect compass



Hidden constants

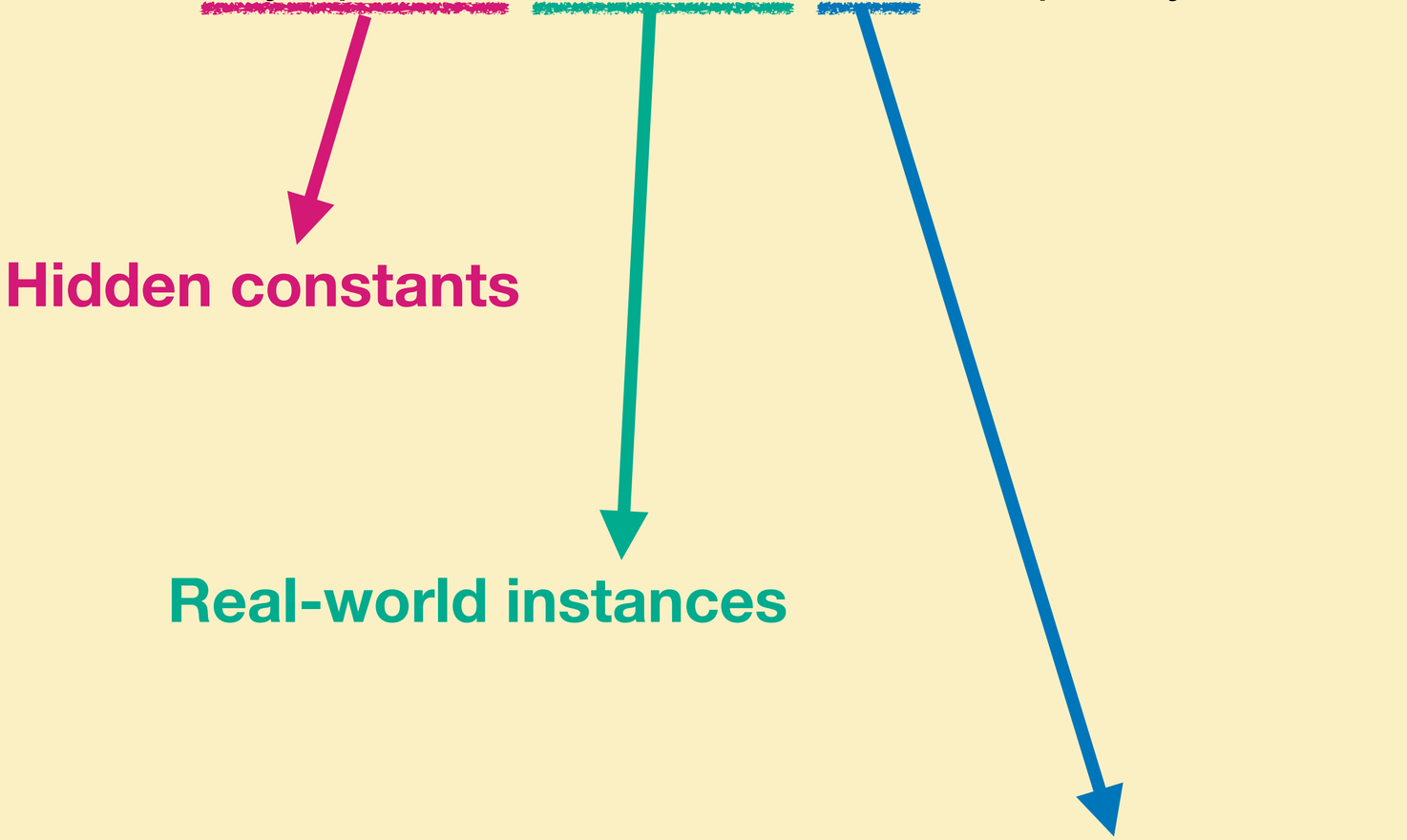
Computational complexity (critics)

Idea. Asymptotical worst-case time complexity in the RAM model is not a perfect compass



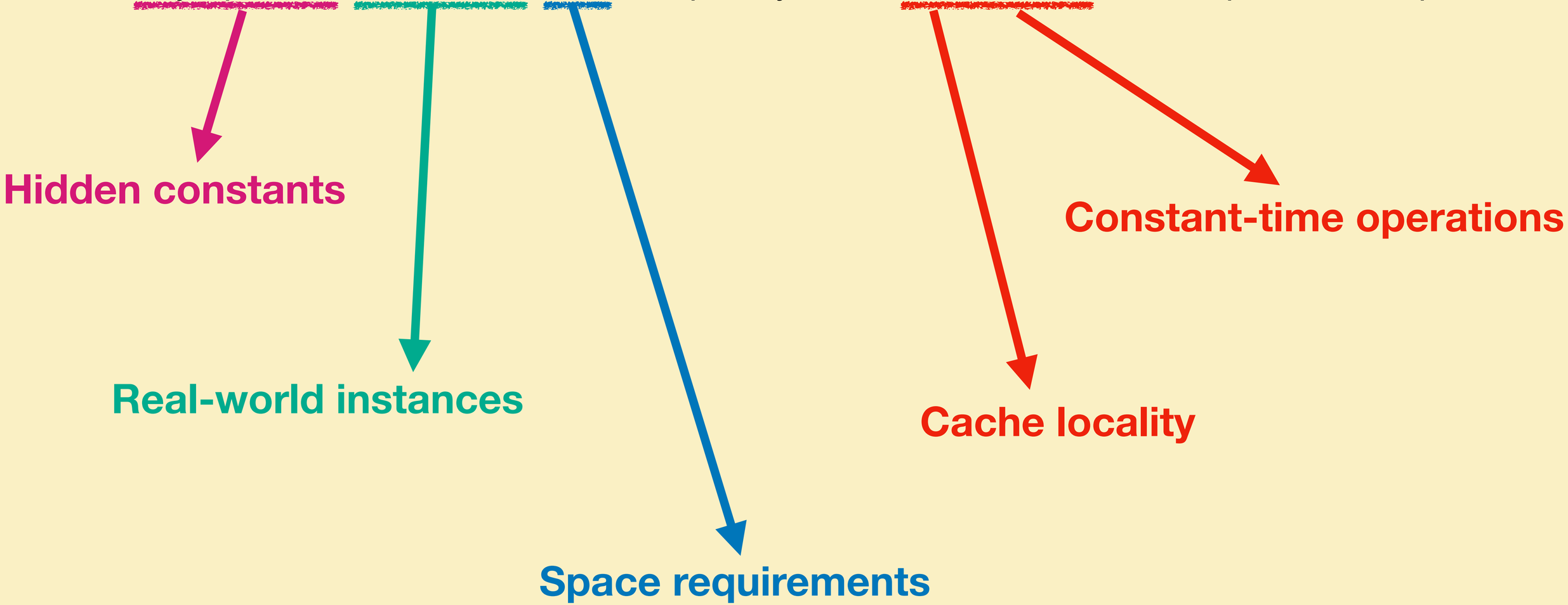
Computational complexity (critics)

Idea. Asymptotical worst-case time complexity in the RAM model is not a perfect compass



Computational complexity (critics)

Idea. Asymptotical worst-case time complexity in the RAM model is not a perfect compass



Black-box thinking

Good reflex. "If I were given a $\mathcal{O}(1)$ algorithm to my problem, would this help?"

Black-box thinking

Good reflex. "If I were given a $\mathcal{O}(1)$ algorithm to my problem, would this help?"

Genome G

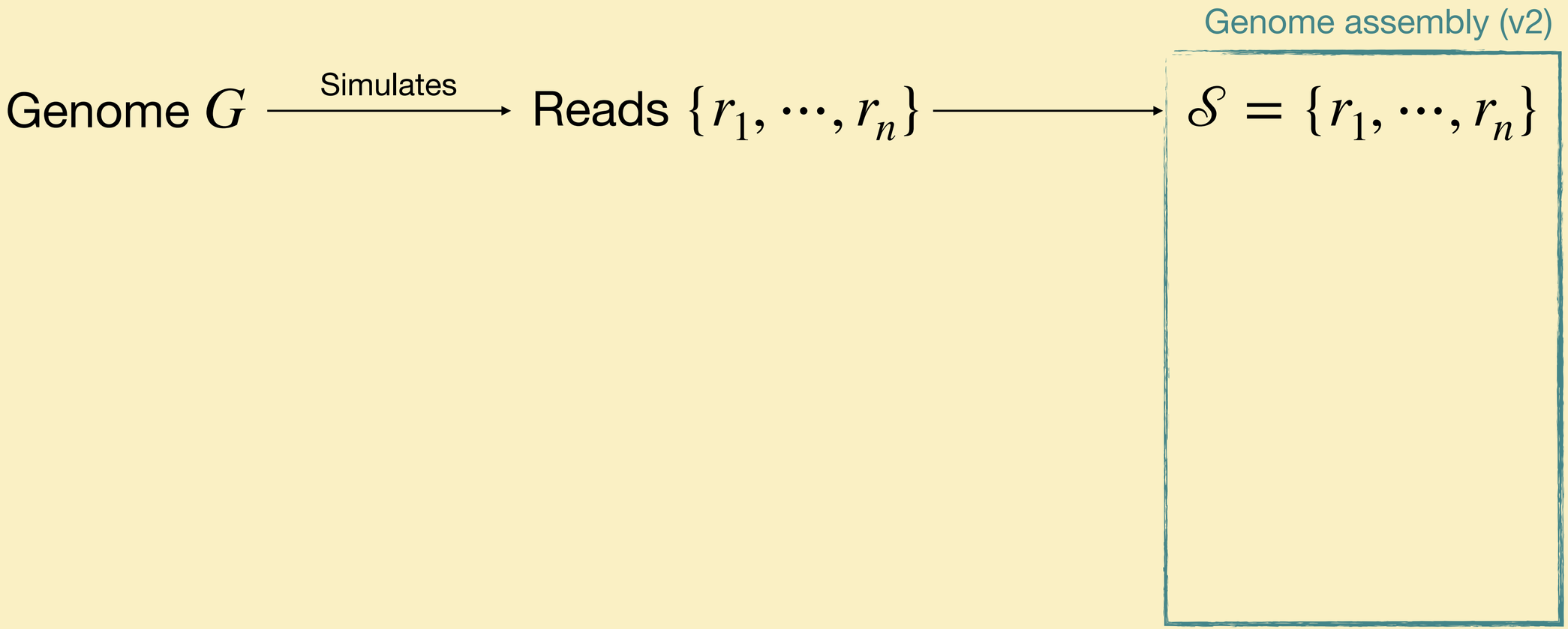
Black-box thinking

Good reflex. "If I were given a $\mathcal{O}(1)$ algorithm to my problem, would this help?"

Genome G $\xrightarrow{\text{Simulates}}$ Reads $\{r_1, \cdots, r_n\}$

Black-box thinking

Good reflex. "If I were given a $\mathcal{O}(1)$ algorithm to my problem, would this help?"



Black-box thinking

Good reflex. "If I were given a $\mathcal{O}(1)$ algorithm to my problem, would this help?"

Genome G —Simulates→ Reads $\{r_1, \cdots, r_n\}$ ⟶

Genome assembly (v2)

$\mathcal{S} = \{r_1, \cdots, r_n\}$

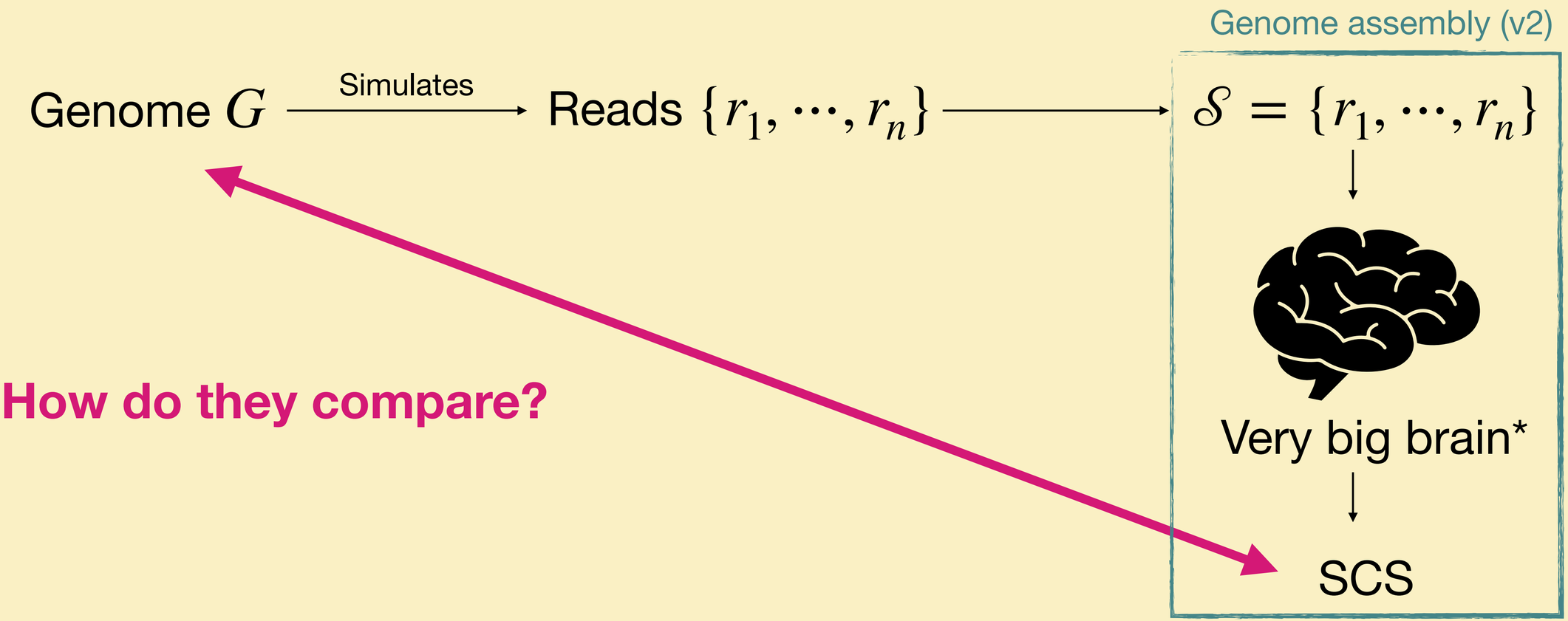
Very big brain*

SCS

*or brute-force on small instance

Black-box thinking

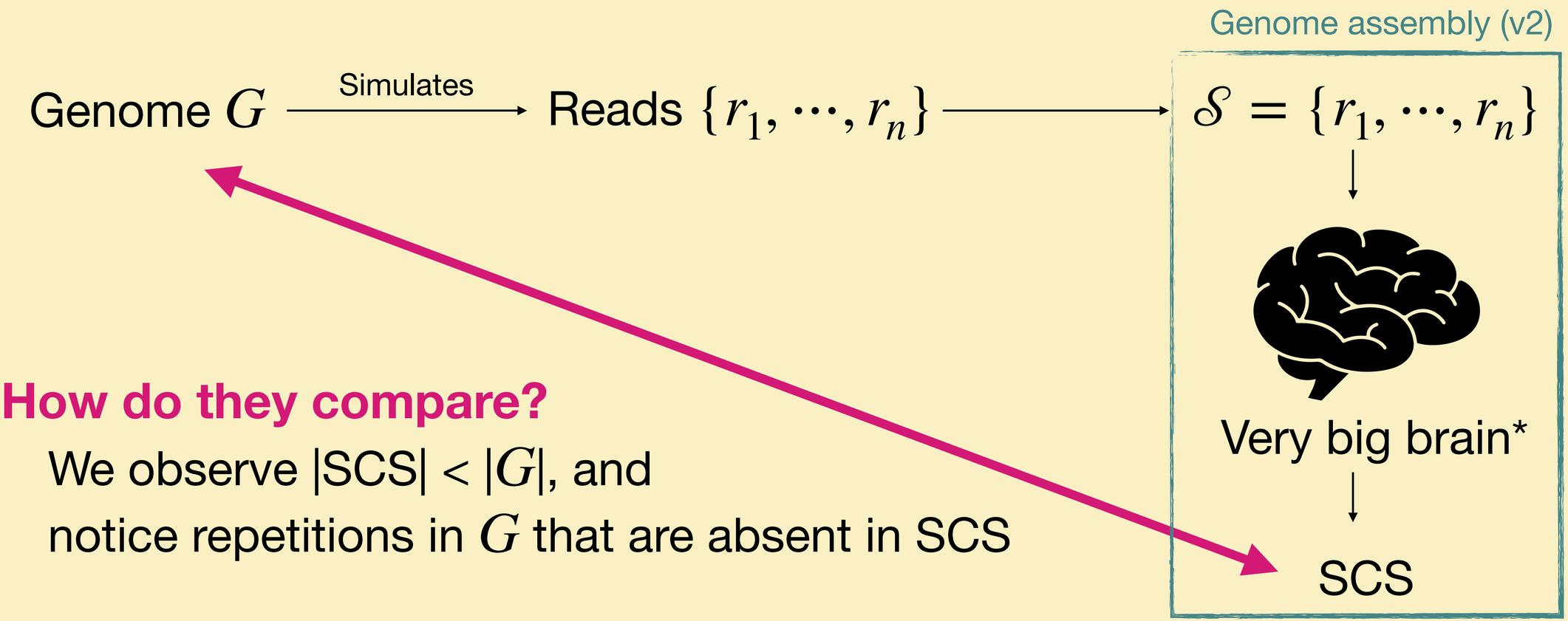
Good reflex. "If I were given a $\mathcal{O}(1)$ algorithm to my problem, would this help?"



*or brute-force on small instance

Black-box thinking

Good reflex. "If I were given a $\mathcal{O}(1)$ algorithm to my problem, would this help?"



How do they compare?

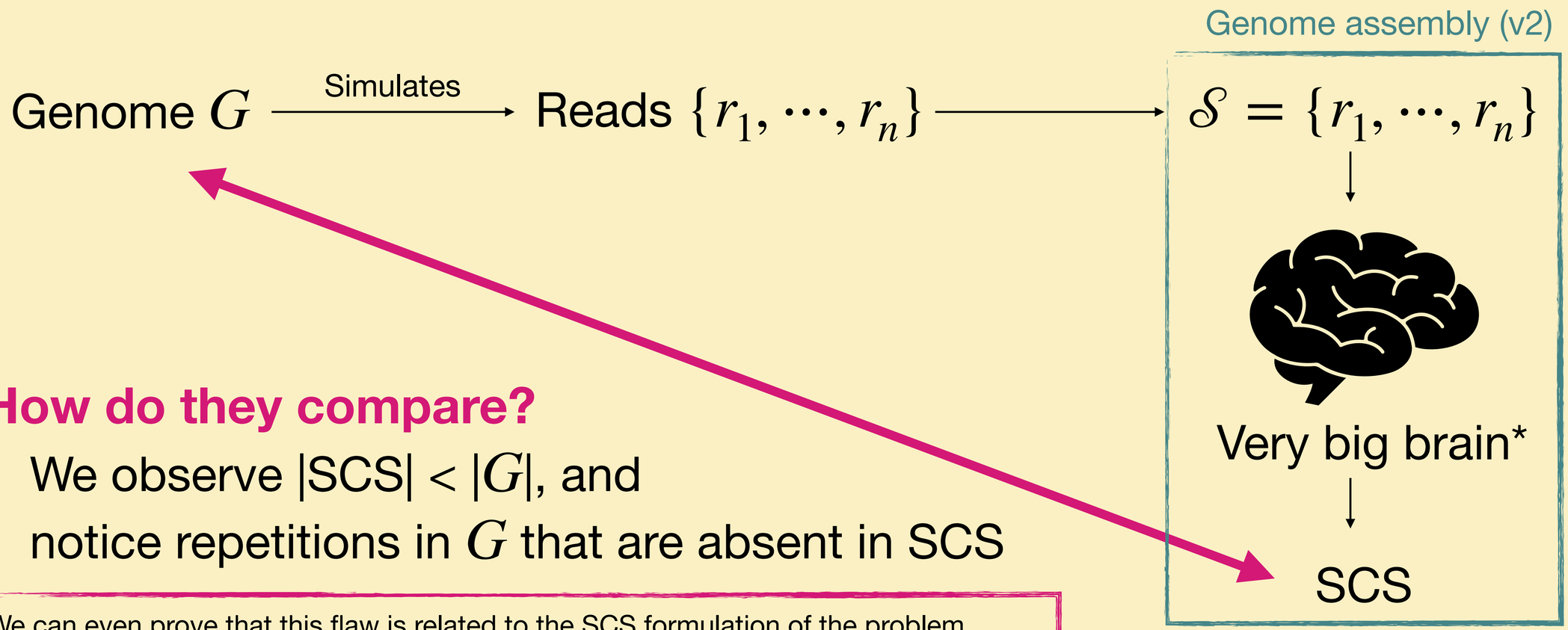
We observe |SCS| < $|G|$, and

notice repetitions in G that are absent in SCS

*or brute-force on small instance

Black-box thinking

Good reflex. "If I were given a $\mathcal{O}(1)$ algorithm to my problem, would this help?"



How do they compare?

We observe |SCS| < $|G|$, and

notice repetitions in G that are absent in SCS

We can even prove that this flaw is related to the SCS formulation of the problem
==> Even if we were able to solve GAv2, we won't solve the original biological question.

*or brute-force on small instance

Refinements and reformulations of the problem

SCS introduces non-existing substrings

$$G_{true} = AATTCCAGCTGATTCCAGT$$

SCS introduces non-existing substrings

$$G_{true} = AATTCCAGCTGATTCCAGT$$

$$\text{Reads}(G_{true}) = \{AAT, ATT, TTC, TCC, CCA, CAG, AGC, GCT, CTG, TGA, GAT\}$$

SCS introduces non-existing substrings

$$G_{true} = AATTCCAGCTGATTCCAGT \qquad SCS = AATTCCAGCTGATAGT$$

$$\text{Reads}(G_{true}) = \{AAT, ATT, TTC, TCC, CCA, CAG, AGC, GCT, CTG, TGA, GAT\}$$

SCS introduces non-existing substrings

$$G_{true} = AATTCCAGCTGATTCCAGT \qquad SCS = AATTCCAGCTGATAGT$$

$$\text{Reads}(G_{true}) = \{AAT, ATT, TTC, TCC, CCA, CAG, AGC, GCT, CTG, TGA, GAT\}$$

SCS introduces non-existing substrings

$$G_{true} = AATTCCAGCTGATTCCAGT \qquad SCS = AATTCCAGCTGATAGT$$

$$\text{Reads}(G_{true}) = \{AAT, ATT, TTC, TCC, CCA, CAG, AGC, GCT, CTG, TGA, GAT\}$$

Idea. Add constraints on the substrings of the solution: all 2-mers of the solution are required to be 2-mers of the genome.

SCS introduces non-existing substrings

$$G_{true} = AATTCCAGCTGATTCCAGT \qquad SCS = AATTCCAGCTGATAGT$$

$$\text{Reads}(G_{true}) = \{AAT, ATT, TTC, TCC, CCA, CAG, AGC, GCT, CTG, TGA, GAT\}$$

Idea. Add constraints on the substrings of the solution: all 2-mers of the solution are required to be 2-mers of the genome.

Definition (Genome assembly, v3).

Input: a collection of strings $\mathcal{S} = \{s_1, \cdots, s_n\}$, an integer k

Output: a shortest string G such that

[1] G is a common superstring of $\mathcal{S}$

[2] $\text{sp}^k(G) \subseteq \text{sp}^k(\mathcal{S})$

SCS introduces non-existing substrings

$$G_{true} = AATTCCAGCTGATTCCAGT \qquad SCS = AATTCCAGCTGATAGT$$

$$\text{Reads}(G_{true}) = \{AAT, ATT, TTC, TCC, CCA, CAG, AGC, GCT, CTG, TGA, GAT\}$$

Idea. Add constraints on the substrings of the solution: all 2-mers of the solution are required to be 2-mers of the genome.

Definition (Genome assembly, v3).

Input: a collection of strings $\mathcal{S} = \{s_1, \cdots, s_n\}$, an integer k

Output: a shortest string G such that

[1] G is a common superstring of $\mathcal{S}$

[2] $\text{sp}^k(G) \subseteq \text{sp}^k(\mathcal{S})$

The genome G_{true} is not an input, so...

SCS introduces non-existing substrings

$$G_{true} = AATTCCAGCTGATTCCAGT \qquad SCS = AATTCCAGCTGATAGT$$

$$\text{Reads}(G_{true}) = \{AAT, ATT, TTC, TCC, CCA, CAG, AGC, GCT, CTG, TGA, GAT\}$$

Idea. Add constraints on the substrings of the solution: all 2-mers of the solution are required to be 2-mers of the genome.

Definition (Genome assembly, v3).

Input: a collection of strings $\mathcal{S} = \{s_1, \cdots, s_n\}$, an integer k

Output: a shortest string G such that

[1] G is a common superstring of $\mathcal{S}$

[2] $\text{sp}^k(G) \subseteq \text{sp}^k(\mathcal{S})$

The genome G_{true} is not an input, so...

Caution. It's possible that a k-mer of the "underlying genome" is not spanned by any read, just because of the fragmentation

=> for well-chosen k (given $|genome|$ and n), this can be made very unlikely

Changing perspective: graphs

Definition (Genome assembly, v3).

Input: a collection of strings $\mathcal{S} = \{s_1, \cdots, s_n\}$, an integer k

Output: a shortest string G such that

[1] G is a common superstring of $\mathcal{S}$

[2] $\text{sp}^k(G) \subseteq \text{sp}^k(\mathcal{S})$

Changing perspective: graphs

Definition (Genome assembly, v3).

Input: a collection of strings $\mathcal{S} = \{s_1, \cdots, s_n\}$, an integer k

Output: a shortest string G such that

[1] G is a common superstring of $\mathcal{S}$

[2] $\mathrm{sp}^k(G) \subseteq \mathrm{sp}^k(\mathcal{S})$

Definition (de Bruijn graph, combinatorics). The dBG of order k is the graph whose vertices are the words of length k (over some alphabet) and u->v iff the $(k-1)$-suffix of u is the $(k-1)$-prefix of v.

Changing perspective: graphs

Definition (Genome assembly, v3).

Input: a collection of strings $\mathcal{S} = \{s_1, \cdots, s_n\}$, an integer k

Output: a shortest string G such that

[1] G is a common superstring of $\mathcal{S}$

[2] $\mathrm{sp}^k(G) \subseteq \mathrm{sp}^k(\mathcal{S})$

Definition (de Bruijn graph, combinatorics). The dBG of order k is the graph whose vertices are the words of length k (over some alphabet) and u->v iff the $(k-1)$-suffix of u is the $(k-1)$-prefix of v.

Definition (de Bruijn graph, bioinformatics). The (node) dBG of order k of a collection of strings is the graph whose vertices are the k-mers of these strings and u->v iff the $(k-1)$-suffix of u is the $(k-1)$-prefix of v.

Changing perspective: graphs

Definition (Genome assembly, v3).

Input: a collection of strings $\mathcal{S} = \{s_1, \cdots, s_n\}$, an integer k

Output: a shortest string G such that

[1] G is a common superstring of $\mathcal{S}$

[2] $\mathrm{sp}^k(G) \subseteq \mathrm{sp}^k(\mathcal{S})$

Definition (de Bruijn graph, combinatorics). The dBG of order k is the graph whose vertices are the words of length k (over some alphabet) and u->v iff the $(k-1)$-suffix of u is the $(k-1)$-prefix of v.

Definition (de Bruijn graph, bioinformatics). The (node) dBG of order k of a collection of strings is the graph whose vertices are the k-mers of these strings and u->v iff the $(k-1)$-suffix of u is the $(k-1)$-prefix of v.

Example. $G_{true} = AATTCCAGCTGATTCCAGT$

Changing perspective: graphs

Definition (Genome assembly, v3).

Input: a collection of strings $\mathcal{S} = \{s_1, \cdots, s_n\}$, an integer k

Output: a shortest string G such that

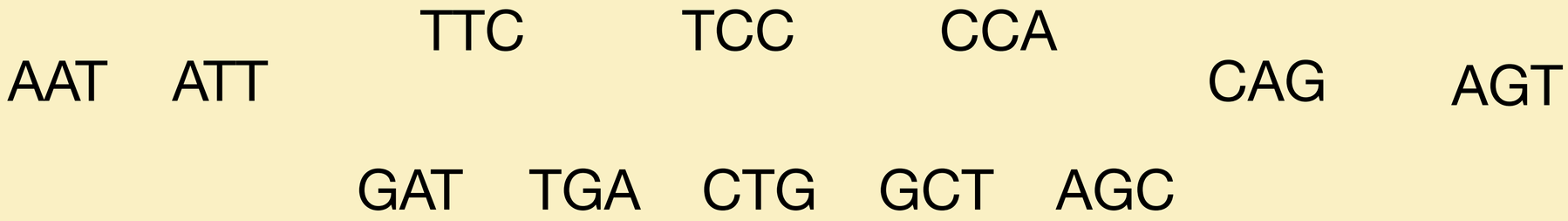
[1] G is a common superstring of $\mathcal{S}$

[2] $\text{sp}^k(G) \subseteq \text{sp}^k(\mathcal{S})$

Definition (de Bruijn graph, combinatorics). The dBG of order k is the graph whose vertices are the words of length k (over some alphabet) and u->v iff the $(k-1)$-suffix of u is the $(k-1)$-prefix of v.

Definition (de Bruijn graph, bioinformatics). The (node) dBG of order k of a collection of strings is the graph whose vertices are the k-mers of these strings and u->v iff the $(k-1)$-suffix of u is the $(k-1)$-prefix of v.

Example. $G_{true} = AATTCCAGCTGATTCCAGT$



Changing perspective: graphs

Definition (Genome assembly, v3).

Input: a collection of strings $\mathcal{S} = \{s_1, \cdots, s_n\}$, an integer k

Output: a shortest string G such that

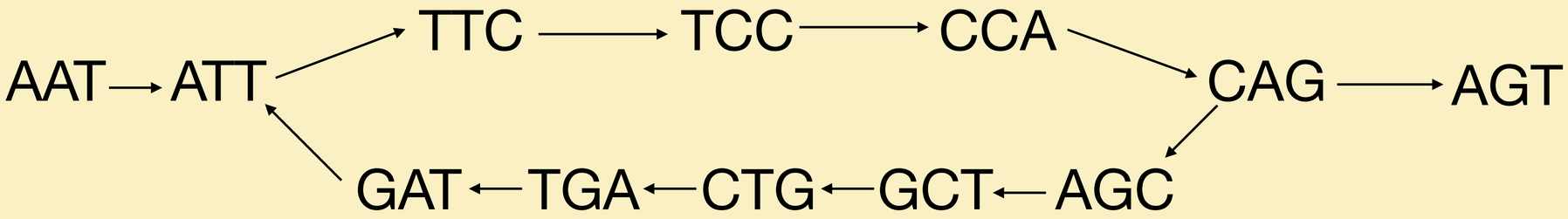
[1] G is a common superstring of $\mathcal{S}$

[2] $\mathrm{sp}^k(G) \subseteq \mathrm{sp}^k(\mathcal{S})$

Definition (de Bruijn graph, combinatorics). The dBG of order k is the graph whose vertices are the words of length k (over some alphabet) and u->v iff the $(k-1)$-suffix of u is the $(k-1)$-prefix of v.

Definition (de Bruijn graph, bioinformatics). The (node) dBG of order k of a collection of strings is the graph whose vertices are the k-mers of these strings and u->v iff the $(k-1)$-suffix of u is the $(k-1)$-prefix of v.

Example. $G_{true} = AATTCCAGCTGATTCCAGT$



Changing perspective: graphs

Definition (Genome assembly, v4).

Input: a collection of strings $\mathcal{S} = \{s_1, \cdots, s_n\}$, an integer k

Output: a shortest string G such that

[1] G is a common superstring of $\mathcal{S}$

[2] $\mathrm{sp}^k(G) \subseteq \mathrm{sp}^k(\mathcal{S})$

Definition (Genome assembly, v5).

Input: a collection of strings $\mathcal{S} = \{s_1, \cdots, s_n\}$, an integer k

Output: The string spelt by an Hamiltonian path in the de Bruijn graph of order k of $\mathcal{S}$

Changing perspective: graphs

Definition (Genome assembly, v4).

Input: a collection of strings $\mathcal{S} = \{s_1, \cdots, s_n\}$, an integer k

Output: a shortest string G such that

[1] G is a common superstring of $\mathcal{S}$

[2] $\mathrm{sp}^k(G) \subseteq \mathrm{sp}^k(\mathcal{S})$

Definition (Genome assembly, v5).

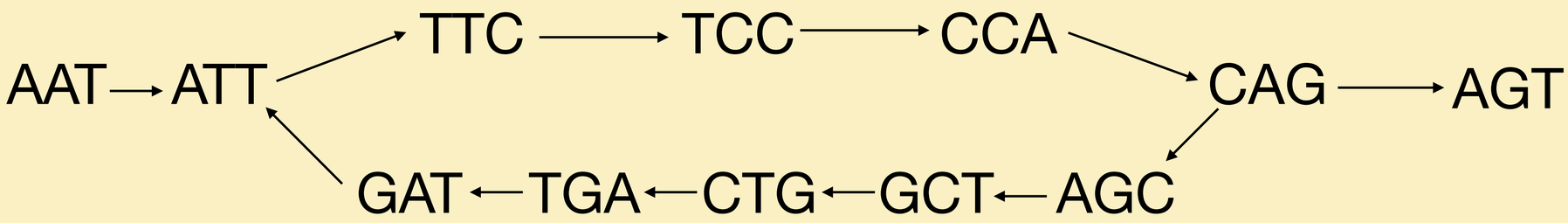
Input: a collection of strings $\mathcal{S} = \{s_1, \cdots, s_n\}$, an integer k

Output: The string spelt by an Hamiltonian path in the de Bruijn graph of order k of $\mathcal{S}$

Looks NP-hard...

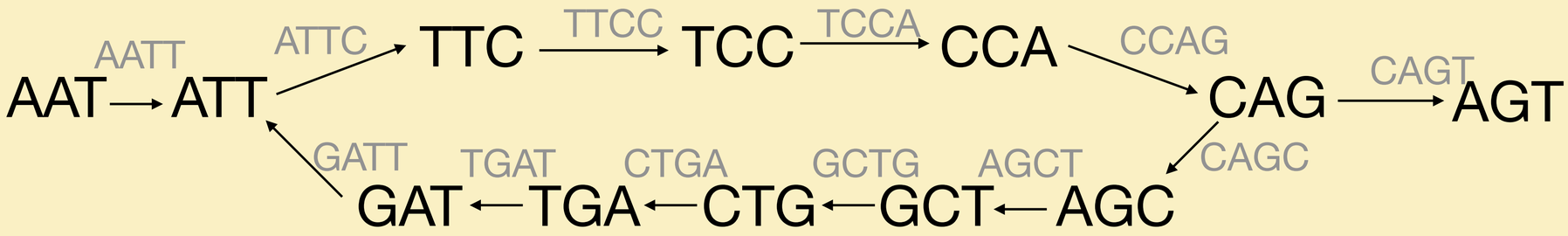
The line graph of dBG-k is dBG-(k+1)

dBG-3.



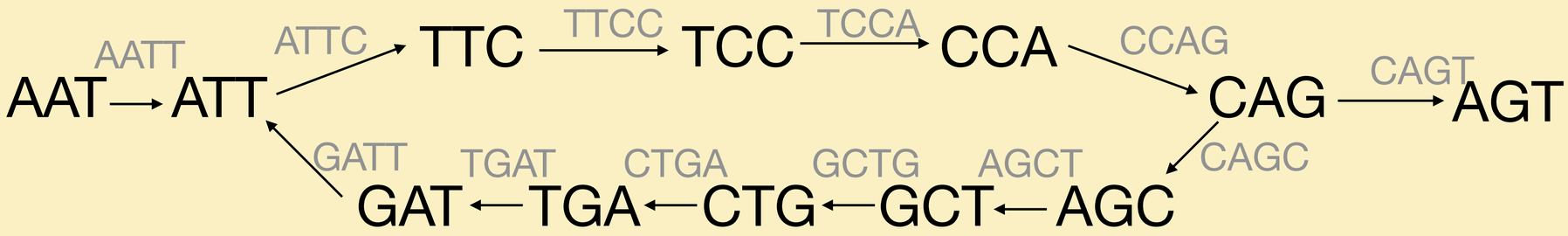
The line graph of dBG-k is dBG-(k+1)

dBG-3.



The line graph of dBG-k is dBG-(k+1)

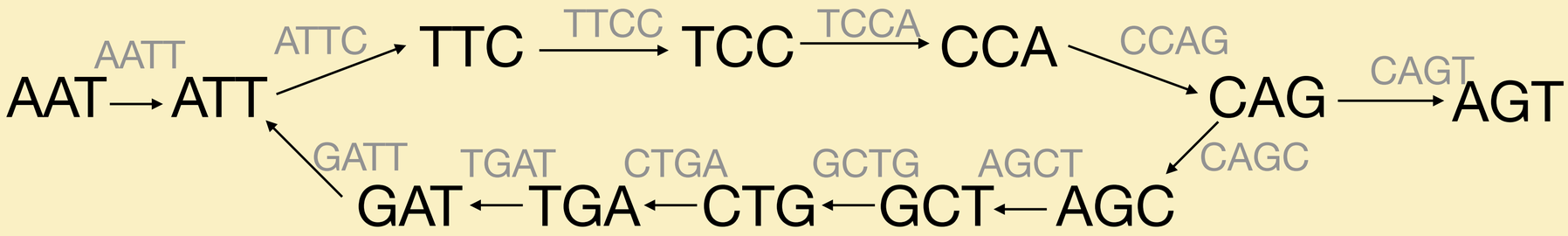
dBG-3.



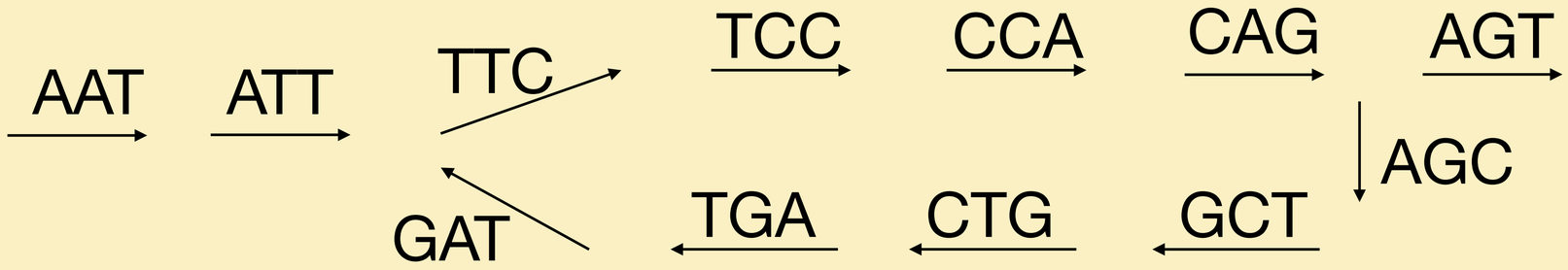
Motivation. If we were interested in linking 4-mers, then it would be an Eulerian path problem => tractable

The line graph of dBG-k is dBG-(k+1)

dBG-3.

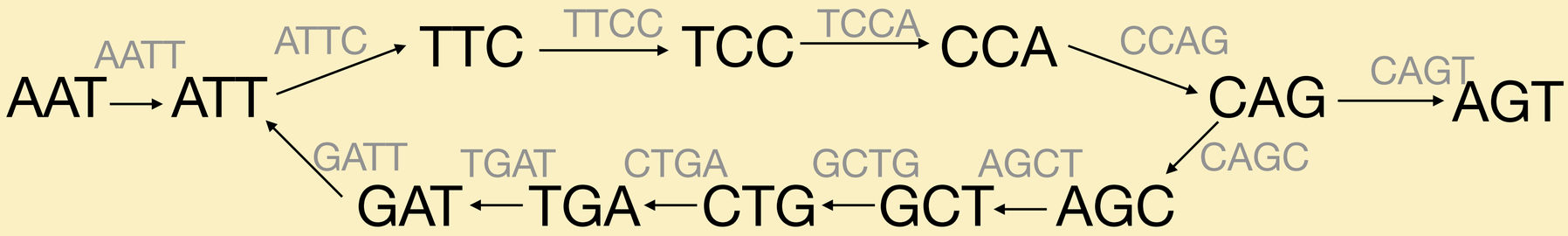


Motivation. If we were interested in linking 4-mers, then it would be an Eulerian path problem => tractable

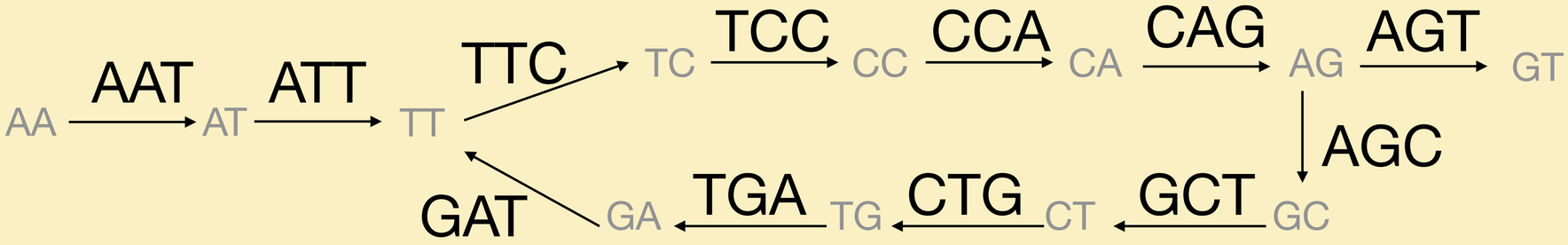


The line graph of dBG-k is dBG-(k+1)

dBG-3.

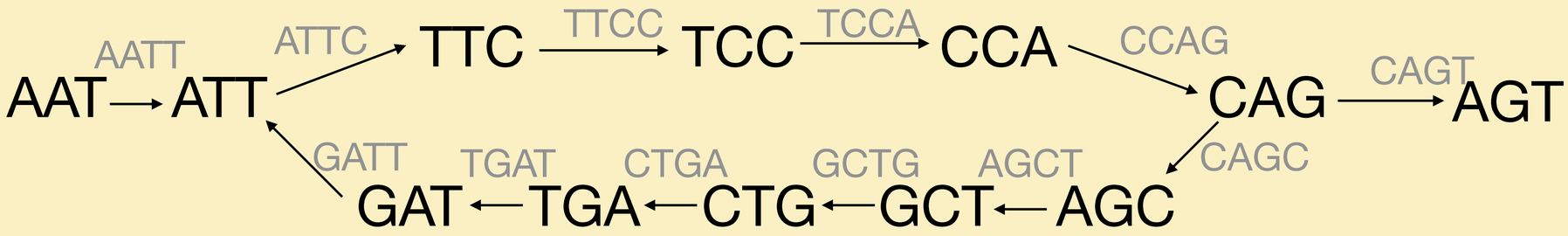


Motivation. If we were interested in linking 4-mers, then it would be an Eulerian path problem => tractable

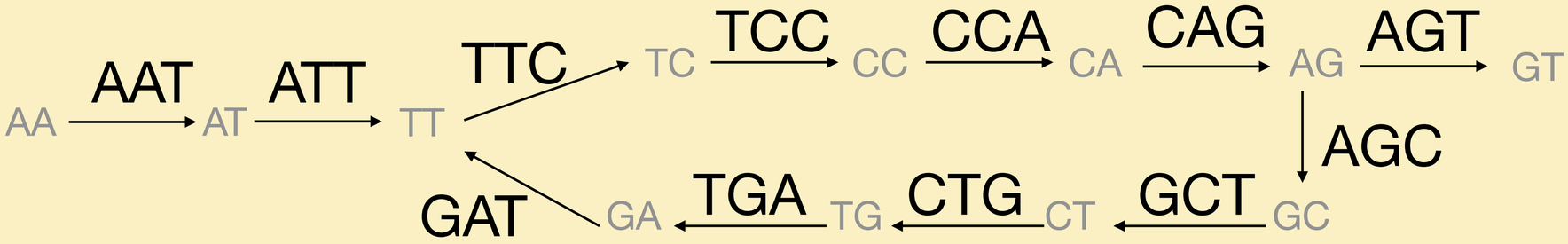


The line graph of dBG-k is dBG-(k+1)

dBG-3.



Motivation. If we were interested in linking 4-mers, then it would be an Eulerian path problem => tractable



Definition (Genome assembly, v6).

Tractable!

Input: a collection of strings $\mathcal{S} = \{s_1, \cdots, s_n\}$, an integer k

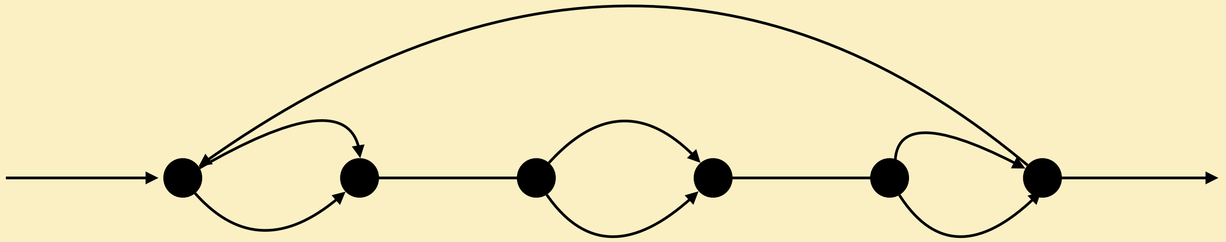
Output: The string spelt by an Eulerian path in the (edge) de Bruijn graph of order $(k-1)$ of $\mathcal{S}$

The impact of branching nodes

Definition (Genome assembly, v6).

Input: a collection of strings $\mathcal{S} = \{s_1, \cdots, s_n\}$, an integer k

Output: The string spelt by an Eulerian path in the (edge) de Bruijn graph of order $(k-1)$ of $\mathcal{S}$

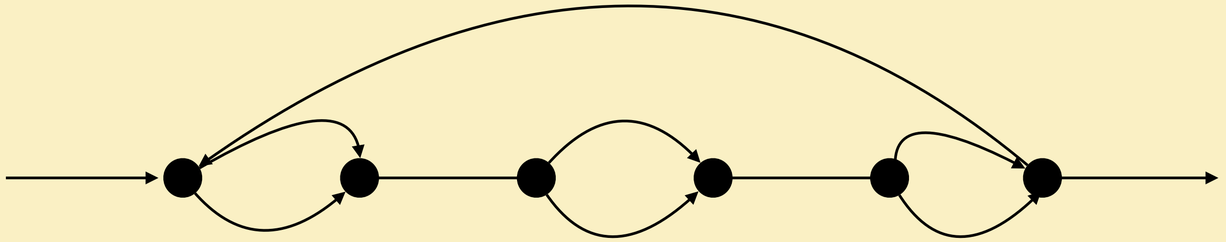


The impact of branching nodes

Definition (Genome assembly, v6).

Input: a collection of strings $\mathcal{S} = \{s_1, \cdots, s_n\}$, an integer k

Output: The string spelt by an Eulerian path in the (edge) de Bruijn graph of order $(k-1)$ of $\mathcal{S}$



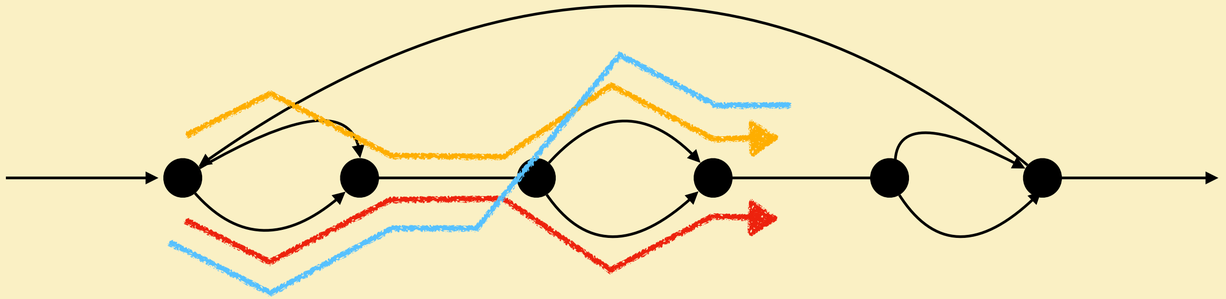
[1] Can create an exponential number of Eulerian paths

The impact of branching nodes

Definition (Genome assembly, v6).

Input: a collection of strings $\mathcal{S} = \{s_1, \cdots, s_n\}$, an integer k

Output: The string spelt by an Eulerian path in the (edge) de Bruijn graph of order $(k-1)$ of $\mathcal{S}$



[1] Can create an exponential number of Eulerian paths

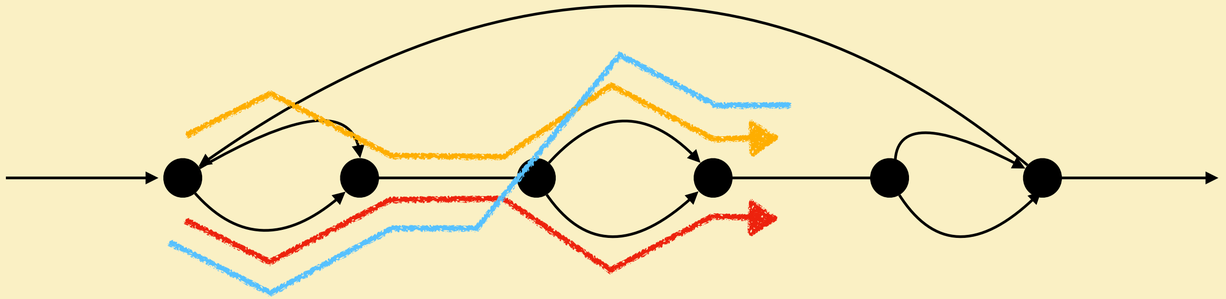
[2] Can create substrings not presents in reads #,# (eg. chimera #)

The impact of branching nodes

Definition (Genome assembly, v6).

Input: a collection of strings $\mathcal{S} = \{s_1, \cdots, s_n\}$, an integer k

Output: The string spelt by an Eulerian path in the (edge) de Bruijn graph of order $(k-1)$ of $\mathcal{S}$



[1] Can create an exponential number of Eulerian paths

[2] Can create substrings not presents in reads #,# (eg. chimera #)

No matter what we do, if the genome is repetitive, we cannot recover it!

A more reasonable problem

A more reasonable problem

Definition (Genome assembly, v7).

Input: a collection of strings $\mathcal{S} = \{s_1, \cdots, s_n\}$, an integer k

Output: The collection of string spelt by non-branching paths in the (edge) de Bruijn graph of order $(k-1)$ of $\mathcal{S}$

A more reasonable problem

Definition (Genome assembly, v7).

Input: a collection of strings $\mathcal{S} = \{s_1, \cdots, s_n\}$, an integer k

Output: The collection of string spelt by non-branching paths in the (edge) de Bruijn graph of order $(k-1)$ of $\mathcal{S}$

Definition (unitig). A path in the dBG is a unitig if

1. all its vertices except the first one have one incoming edge
2. all its vertices except the last one have one outcoming edge

A more reasonable problem

Definition (Genome assembly, v7).

Input: a collection of strings $\mathcal{S} = \{s_1, \cdots, s_n\}$, an integer k

Output: The collection of string spelt by non-branching paths in the (edge) de Bruijn graph of order $(k-1)$ of $\mathcal{S}$

Definition (unitig). A path in the dBG is a unitig if

1. all its vertices except the first one have one incoming edge
2. all its vertices except the last one have one outcoming edge

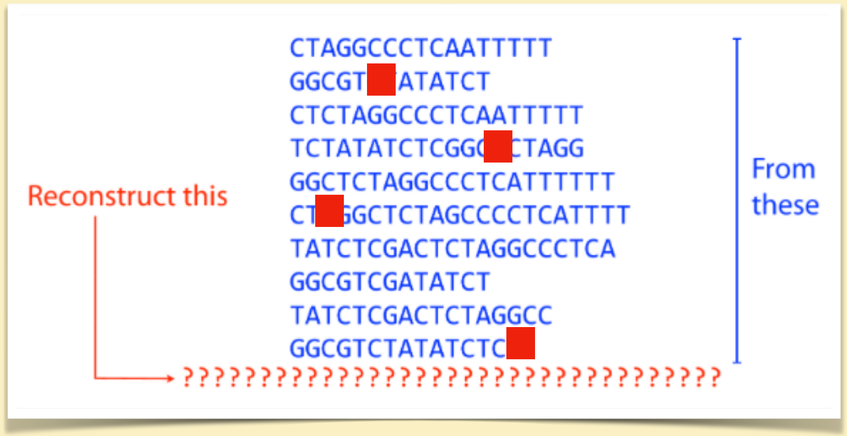
Morally, these are strings that appear as substring in any genome reconstruction based on the dBG

A last word on "usefulness"

(Hidden) assumptions we've made

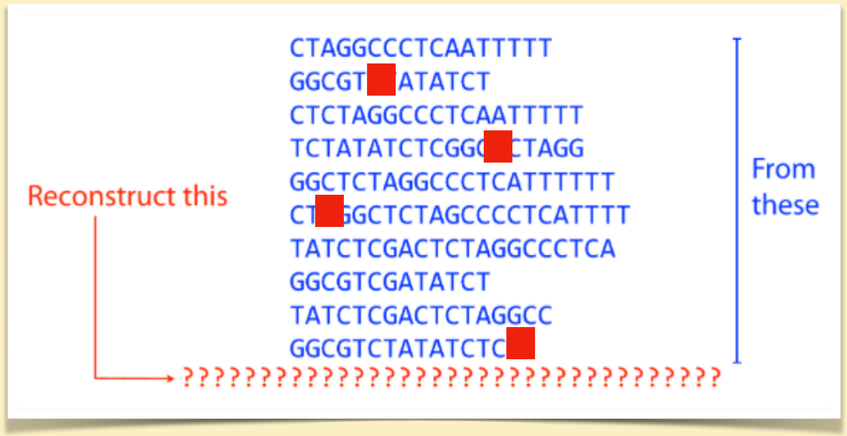
(Hidden) assumptions we've made

No errors in the reads

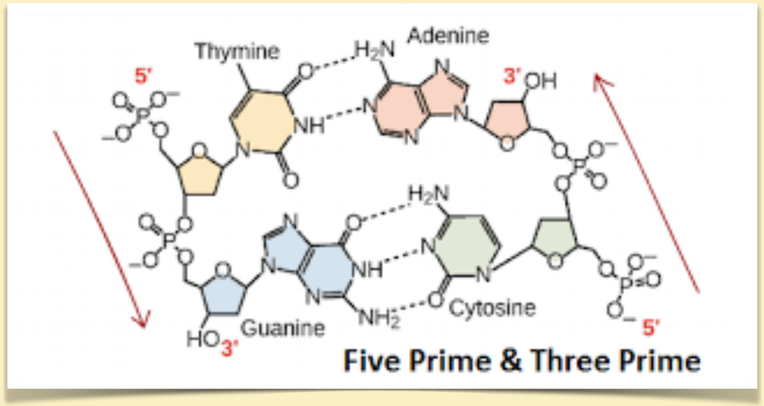


(Hidden) assumptions we've made

No errors in the reads



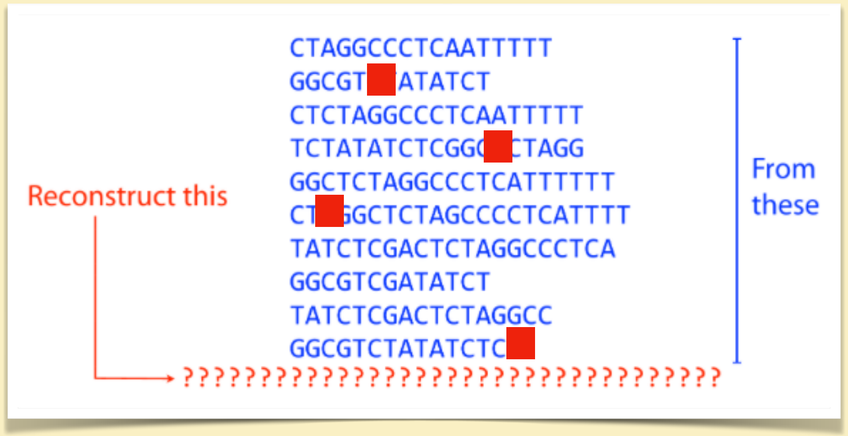
All reads are 5' to 3'



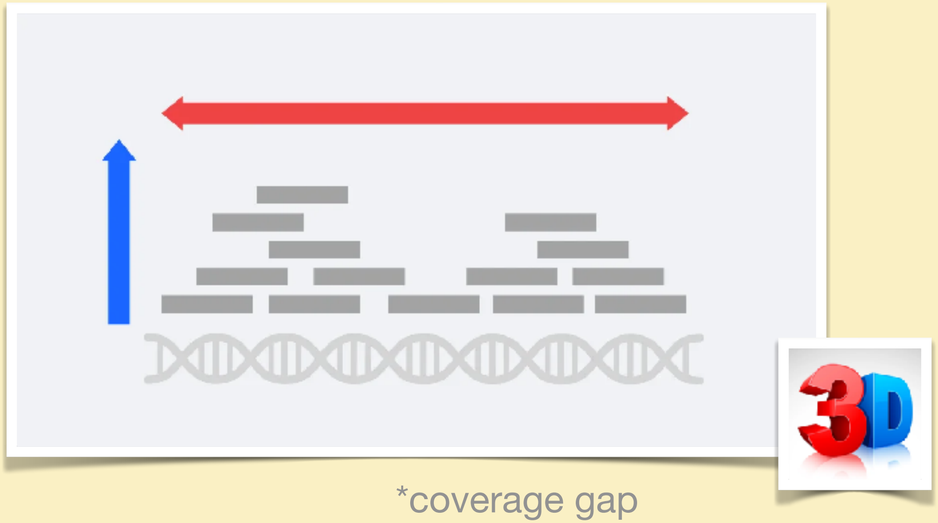
*bidirectional model

(Hidden) assumptions we've made

No errors in the reads

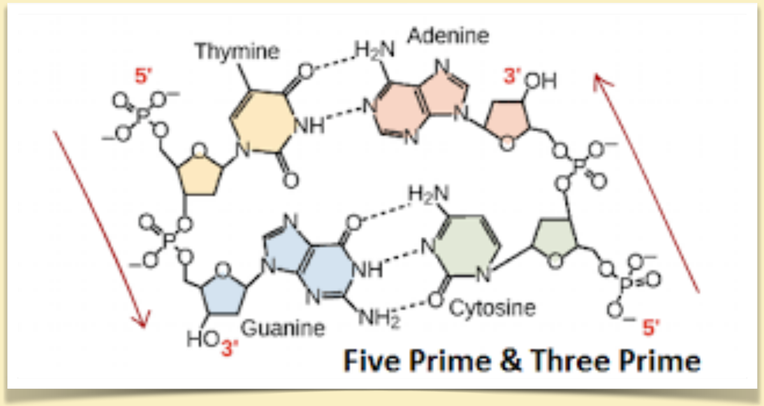


All k-mers are present in the reads



*coverage gap

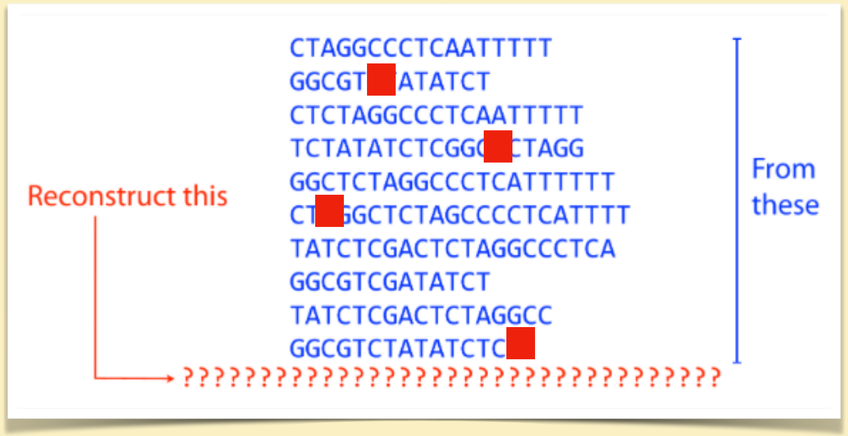
All reads are 5' to 3'



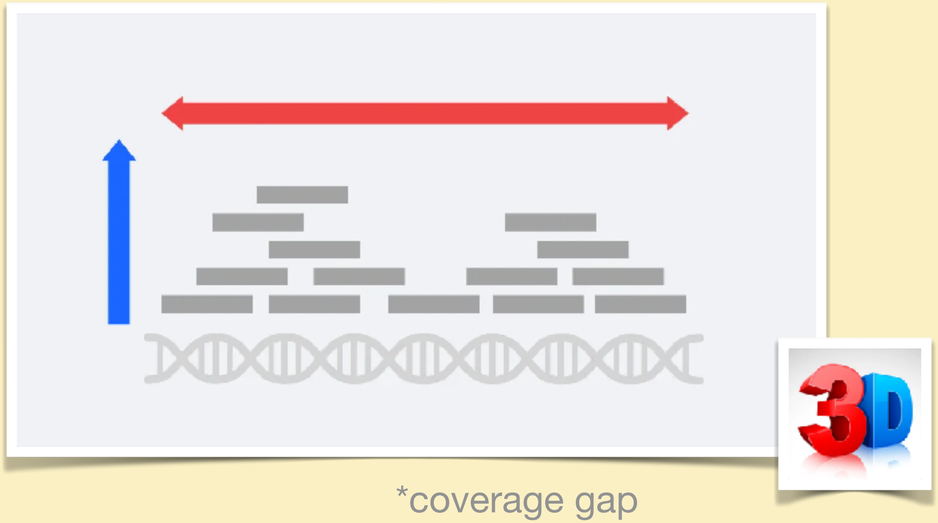
*bidirectional model

(Hidden) assumptions we've made

No errors in the reads

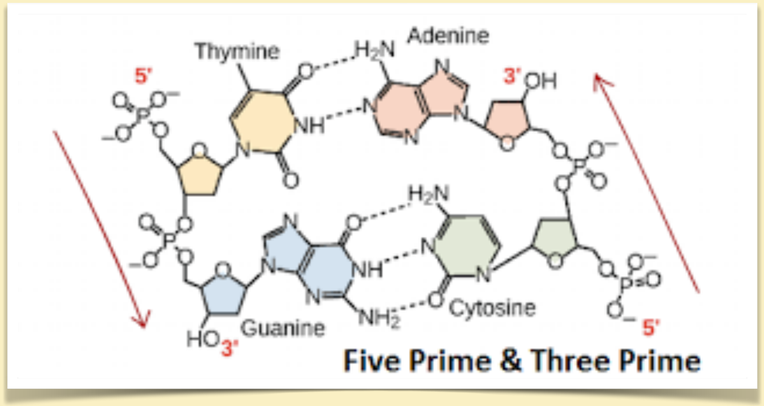


All k-mers are present in the reads



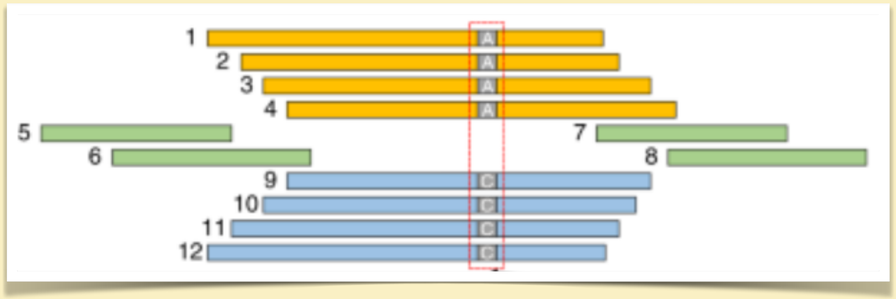
*coverage gap

All reads are 5' to 3'



*bidirectional model

Single haploids



Origin of flaws

Motto. **The ultimate test of a (practical) algorithm is how is performs on real data!**

What if it doesn't work?

Origin of flaws

Motto. **The ultimate test of a (practical) algorithm is how is performs on real data!**

What if it doesn't work?

Finding the origin of flaws.

A. Simulate data under algorithm's assumptions
B. Simulate biological data to the best we understand it
C. Use real data

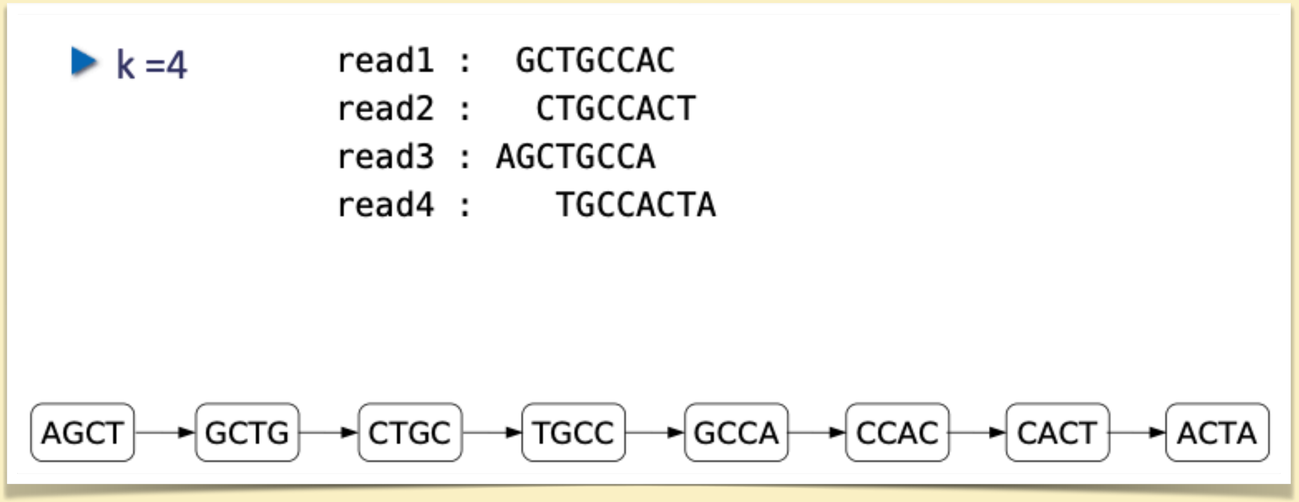
Part II

Solving GAv7 (DNA consensus regions)

De Bruijn graphs

The *de Bruijn* graph

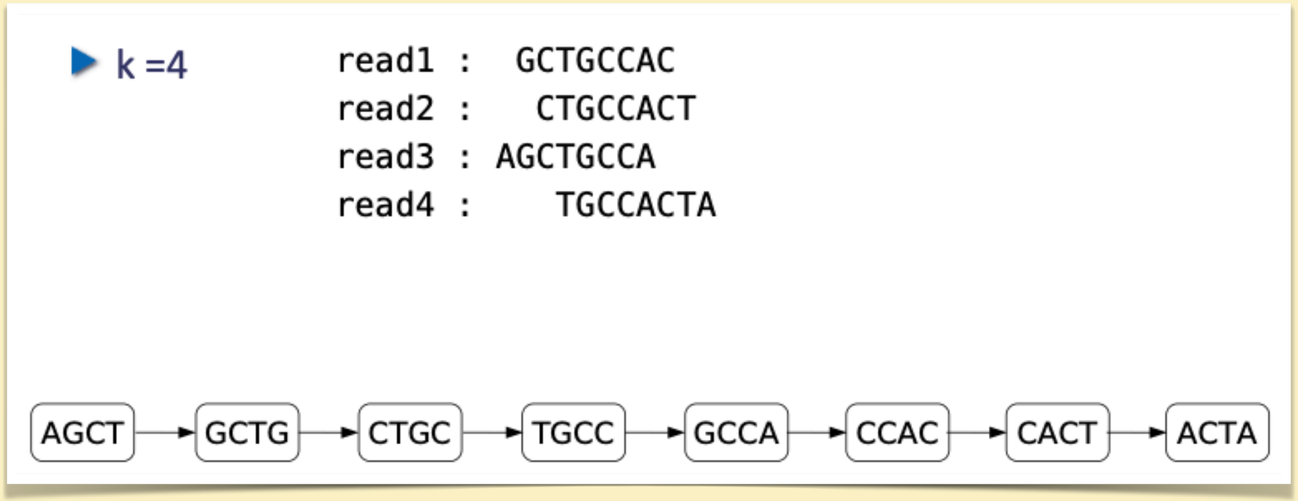
Historical context. Coined in 1946, used in bioinformatics in 1995.



*finding unitigs is easy (compared to HP) -> stick to node-centric

The *de Bruijn* graph

Historical context. Coined in 1946, used in bioinformatics in 1995.



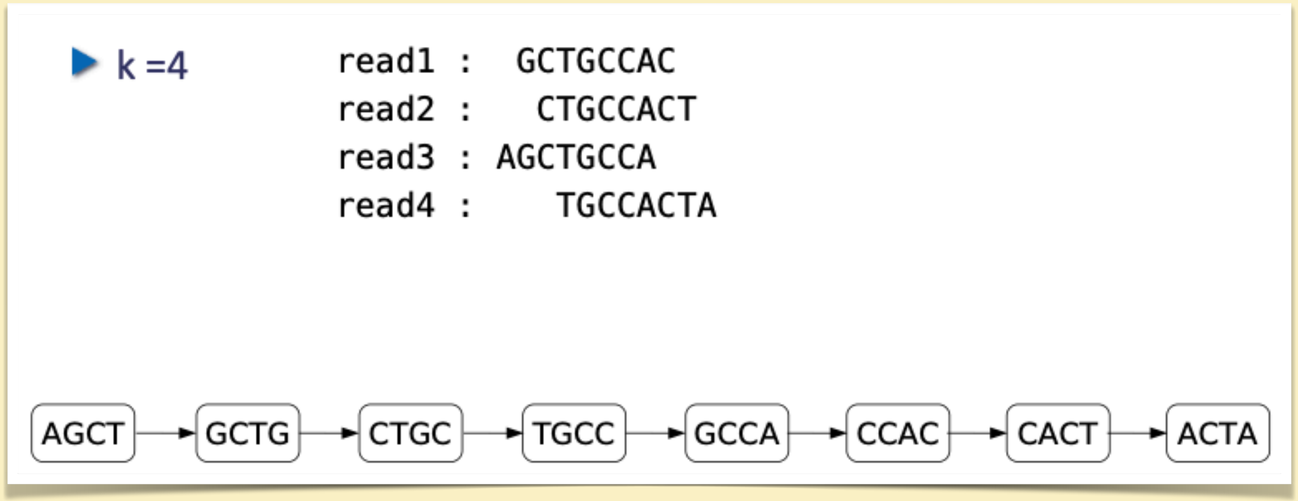
*finding unitigs is easy (compared to HP) -> stick to node-centric

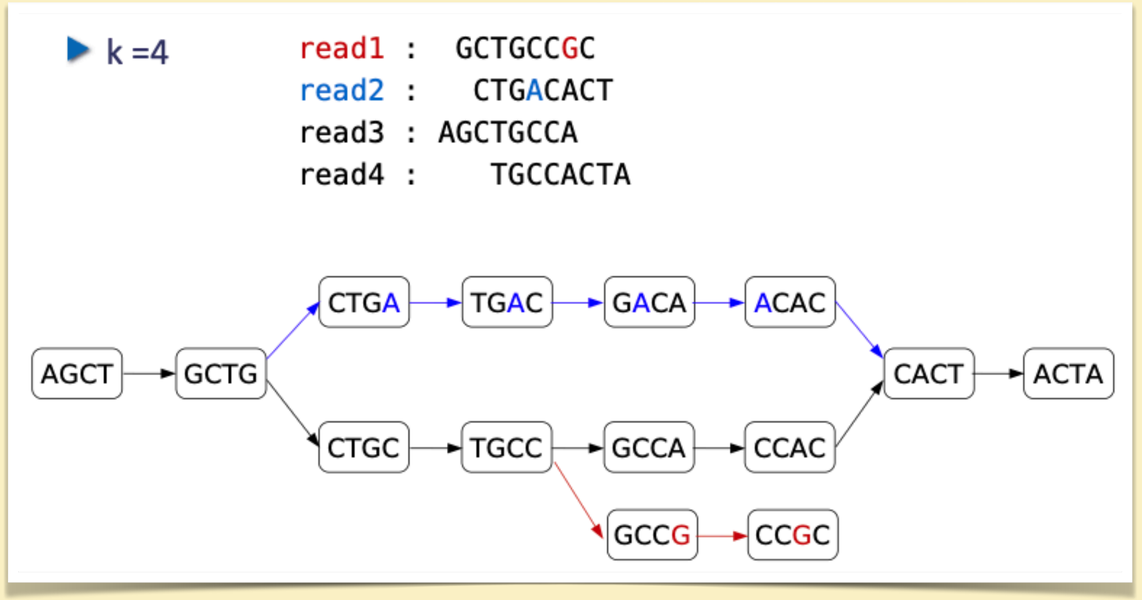
Benefits.

- The graph can be computed efficiently (exact and fixed-length overlaps)
- Roughly $\mathcal{O}(|genome|)$ nodes, doesn't depend on the read coverage

Is there any drawbacks?

Sequencing errors in the *de Bruijn* graph

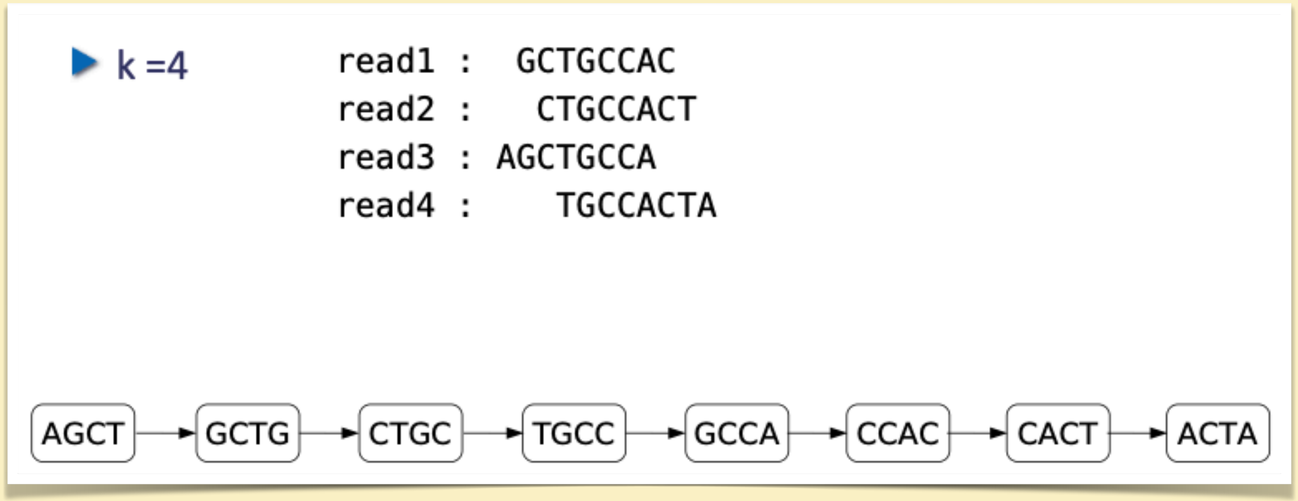


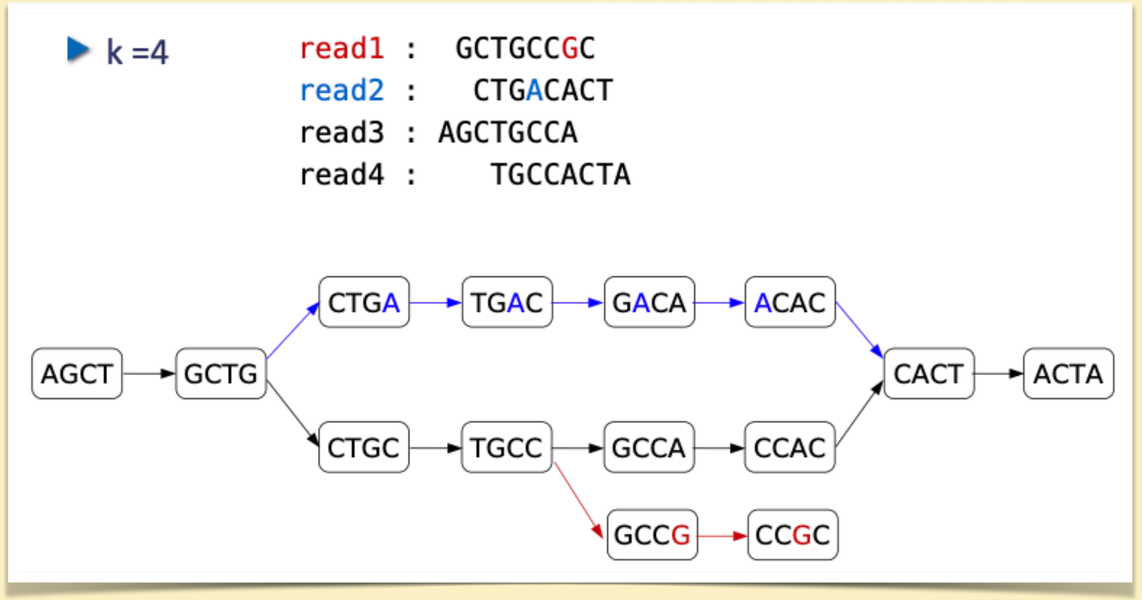


=> Even single nucleotide errors have great impact on the topology

Up to k new vertices, new branching paths

Sequencing errors in the *de Bruijn* graph





=> Even single nucleotide errors have great impact on the topology

Up to k new vertices, new branching paths

=> Challenging task: clean the graph WITHOUT losing SNPs (biological single nucleotide variants)

Topology (candidate error zones), k-mer abundance (decide if error)

The choice of k in the *de Bruijn* graph

As k grows.

The choice of k in the *de Bruijn* graph

As k grows.

- the graph becomes more linear as the k-mer is more likely to appear only once in the genome (except. repetition)

The choice of k in the *de Bruijn* graph

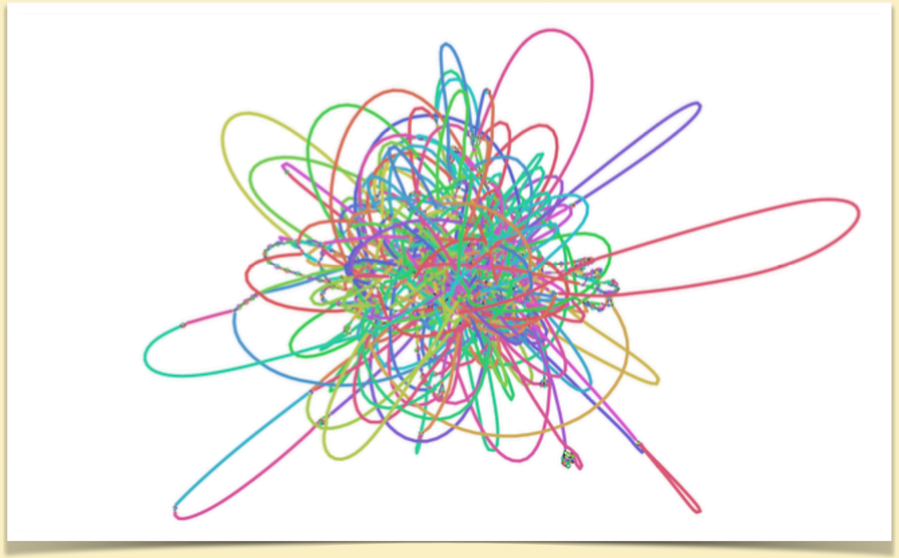
As k grows.

- the graph becomes more linear as the k-mer is more likely to appear only once in the genome (except. repetition)
- the graph becomes disconnected, as some k-mers are not seen in any reads

The choice of k in the *de Bruijn* graph

As k grows.

- the graph becomes more linear as the k-mer is more likely to appear only once in the genome (except. repetition)
- the graph becomes disconnected, as some k-mers are not seen in any reads

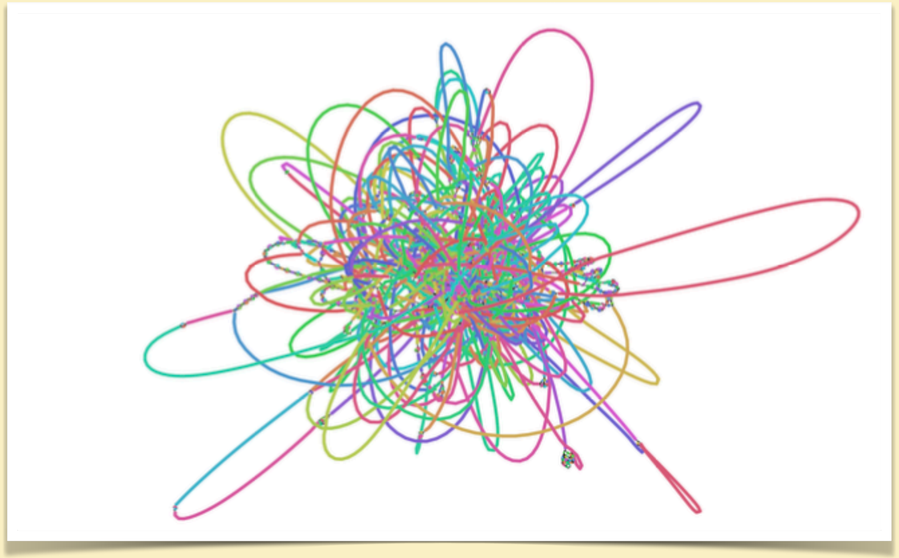


E. Coli, k=31, main component

The choice of k in the *de Bruijn* graph

As k grows.

- the graph becomes more linear as the k-mer is more likely to appear only once in the genome (except. repetition)
- the graph becomes disconnected, as some k-mers are not seen in any reads



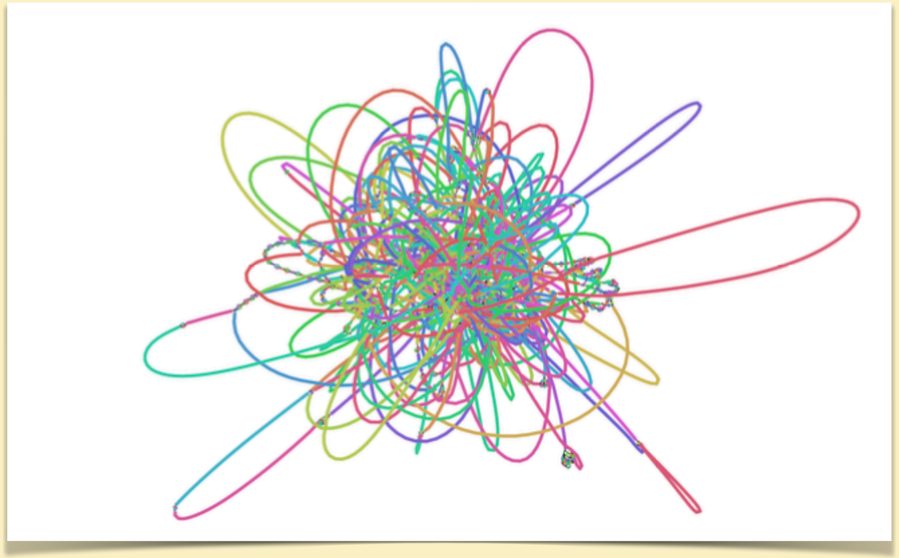
E. Coli, k=31, main component

E. Coli, k=62, main component

The choice of k in the *de Bruijn* graph

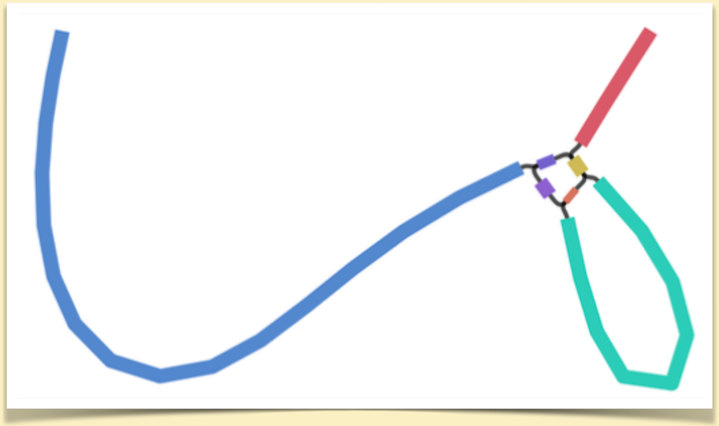
As k grows.

- the graph becomes more linear as the k-mer is more likely to appear only once in the genome (except. repetition)
- the graph becomes disconnected, as some k-mers are not seen in any reads



E. Coli, k=31, main component

E. Coli, k=62, main component



E. Coli, k=2000, main component

Storage requirements

Storing vertices. $\log_2(4) = 2$ bits per k-mer, hence $\approx 2k \cdot |genome|$ bits in total

Storing edges. at least $\log_2(2k \cdot |genome|)$ bits per pointer,
hence $\approx \log_2(2k \cdot |genome|) \cdot 4 \cdot 2k|genome|$ bits in total

Huge!

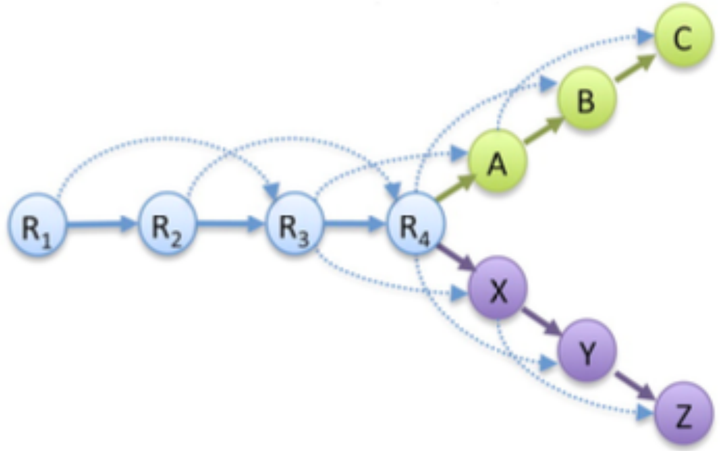
In practice, edges are not stored but computed on the fly (eg. Bloom filter, see later.)

Overlap graphs (overlap-layout-consensus)

A (slightly) different approach

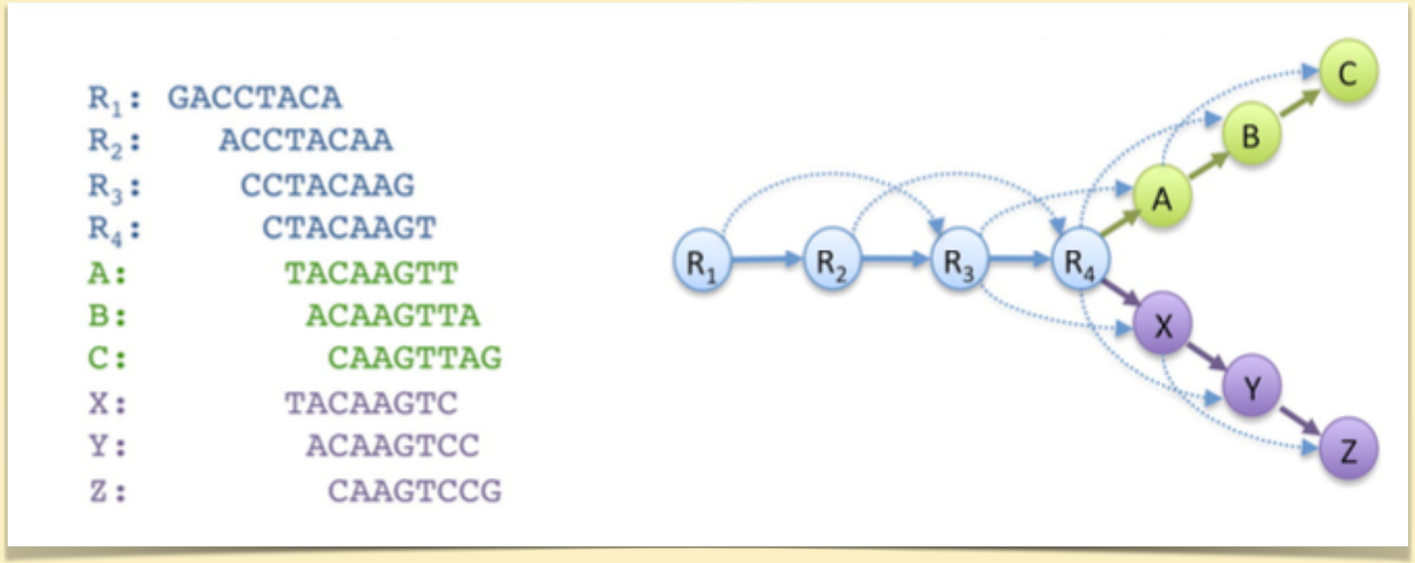
Definition. The **overlap graph** is the graph whose vertices are the reads, and the edges connect any overlapping vertices

```
R1: GACCTACA
R2:    ACCTACAA
R3:     CCTACAAG
R4:      CTACAAGT
A:        TACAAGTT
B:         ACAAGTTA
C:          CAAGTTAG
X:        TACAAGTC
Y:         ACAAGTCC
Z:          CAAGTCCG
```

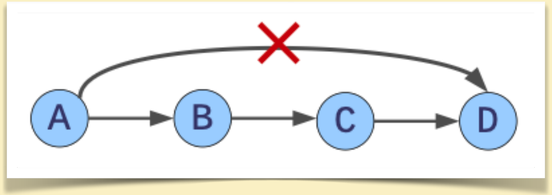


A (slightly) different approach

Definition. The **overlap graph** is the graph whose vertices are the reads, and the edges connect any overlapping vertices



Note. Edges that can be deduced from transitivity are often removed



Vertices (reads) included in others are also removed

Approximate overlap is often used

eg. 2 errors overlap

```
ATTCCTCATCT
||  ||
ACAGCTATTCT
```

More on the overlap graph

Prop. Let N be the number of reads. The graph has N vertices and $\mathcal{O}(N^2)$ edges.

More on the overlap graph

Prop. Let N be the number of reads. The graph has N vertices and $\mathcal{O}(N^2)$ edges.

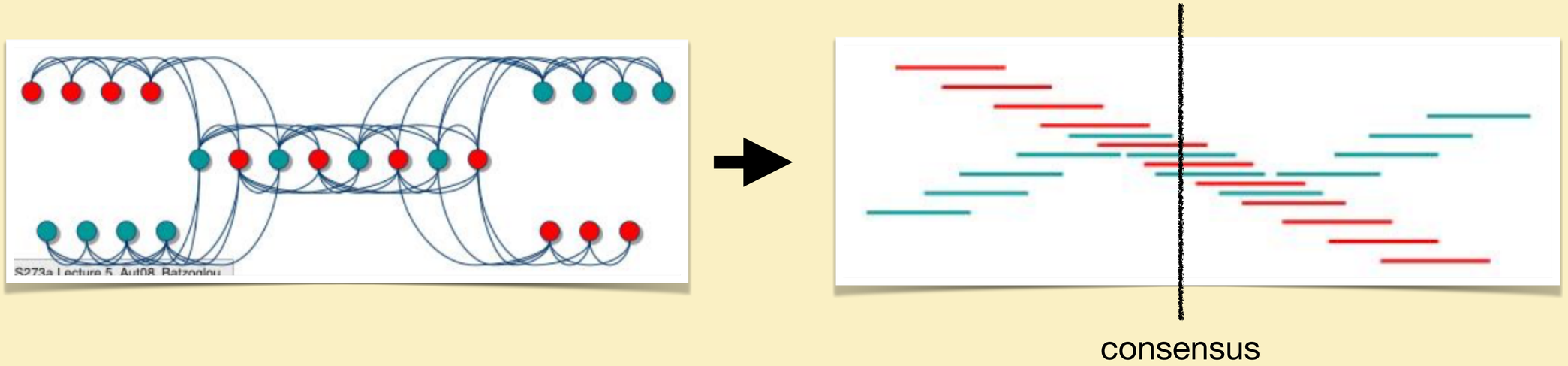
The construction of the graph requires to align the reads against each other => N^2 such alignments

More on the overlap graph

Prop. Let N be the number of reads. The graph has N vertices and $\mathcal{O}(N^2)$ edges.

The construction of the graph requires to align the reads against each other => N^2 such alignments

Correcting reads in the approximate overlap setting.



Part III (next time)

Real-world genome assembly

Homework.

A. For each of the $\{\text{short}, \text{long}\} \times \{\text{high quality}, \text{low quality}\}$ types of reads, what method (dBG/OG) is better suited? Why?

B. Propose (on paper) a complete assembly pipeline, that includes graph-cleaning steps. Make design assumptions explicit and expected flaws as clear as possible. Estimate the space/time complexity of each step.