Computational Aspects in Gene Order Studies

Jens Stoye

AG Genominformatik, Technische Fakultät

Institute of Bioinformatics, Center of Biotechnology

Bielefeld University, Germany

Comparative genomics

Comparative genomics is applied with several goals in mind:

- Comparative gene finding, annotation
- Binding site prediction, phylogenetic footprinting
- Genome alignment
- Correlated expression
- Conserved gene neighborhood
- Rearrangement studies

Comparative genomics “at a higher level”

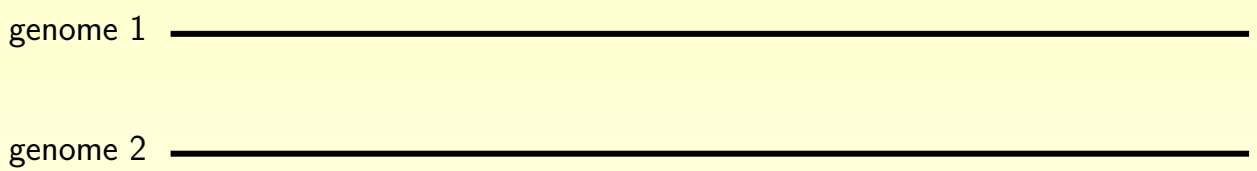
Concentrate on large scale layout of the genomes:

- Study genomes based on their *gene order*.
- Represent genomes by their sequence of genes.

Comparative genomics “at a higher level”

Concentrate on large scale layout of the genomes:

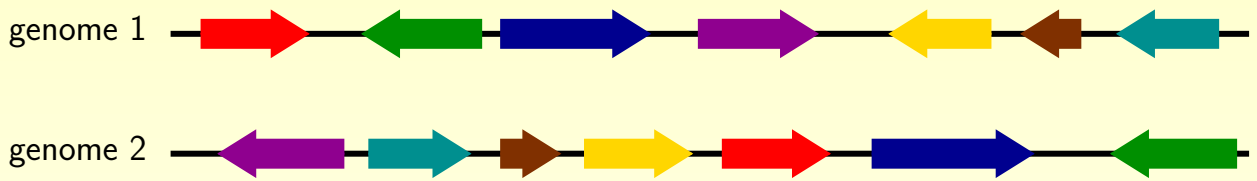
- Study genomes based on their *gene order*.
- Represent genomes by their sequence of genes.



Comparative genomics “at a higher level”

Concentrate on large scale layout of the genomes:

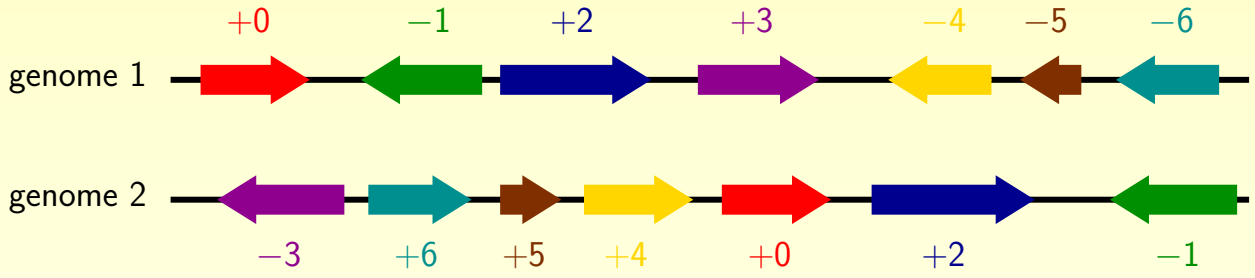
- Study genomes based on their *gene order*.
- Represent genomes by their sequence of genes.



Comparative genomics “at a higher level”

Concentrate on large scale layout of the genomes:

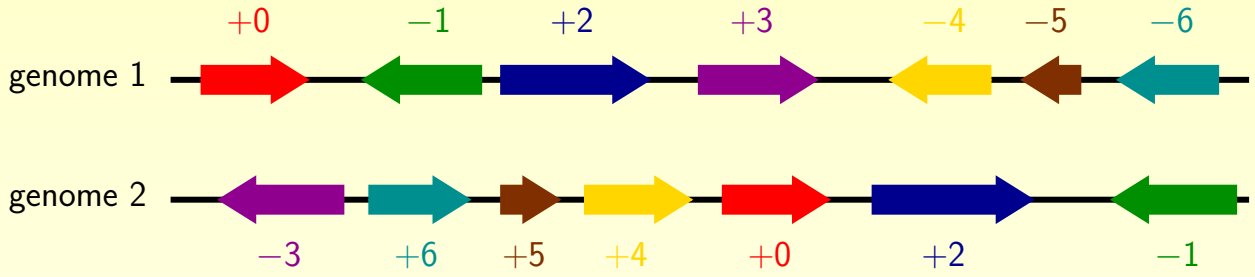
- Study genomes based on their *gene order*.
- Represent genomes by their sequence of genes.



Comparative genomics "at a higher level"

Concentrate on large scale layout of the genomes:

- Study genomes based on their *gene order*.
- Represent genomes by their sequence of genes.



More formally:

- Genes = (signed) elements from the set $N = \{0, \ldots, n\}$.
- Corresponding (*orthologous*) genes get the same number.
- Genomes = (signed) permutations of N.

Protein function prediction

The yeast *S. cerevisiae* was sequenced in 1995.
Still, about 30% of the ORFs in the *MIPS Yeast Database* have no functional annotation.
Functional annotation is time consuming and expensive.

Protein function prediction

The yeast *S. cerevisiae* was sequenced in 1995.
Still, about 30% of the ORFs in the *MIPS Yeast Database* have no functional annotation.
Functional annotation is time consuming and expensive.

- In the lab:
  - Genetical and biochemical analysis
  - Correlated expression

Protein function prediction

The yeast *S. cerevisiae* was sequenced in 1995.
Still, about 30% of the ORFs in the *MIPS Yeast Database* have no functional annotation.
Functional annotation is time consuming and expensive.

- In the lab:
  - Genetical and biochemical analysis
  - Correlated expression
- Homology based:
  - Protein families
  - Functional domains

Protein function prediction

The yeast *S. cerevisiae* was sequenced in 1995.
Still, about 30% of the ORFs in the *MIPS Yeast Database* have no functional annotation.
Functional annotation is time consuming and expensive.

- In the lab:
  - Genetical and biochemical analysis
  - Correlated expression
- Homology based:
  - Protein families
  - Functional domains
- Genome based:
  - Rosetta stone method (gene fusion, domain fusion)
  - Phylogenetic profiles (correlated evolution)
  - Gene order (co-occurrence of genes in genomes)

Protein function prediction

The yeast *S. cerevisiae* was sequenced in 1995.
Still, about 30% of the ORFs in the *MIPS Yeast Database* have no functional annotation.
Functional annotation is time consuming and expensive.

- In the lab:
  - Genetical and biochemical analysis
  - Correlated expression
- Homology based:
  - Protein families
  - Functional domains
- Genome based:
  - Rosetta stone method (gene fusion, domain fusion)
  - Phylogenetic profiles (correlated evolution)
  - Gene order (co-occurrence of genes in genomes)
- Literature based:
  - Natural language processing

Genome-based gene function prediction

Comparative genomics meets functional genomics.

Idea: Genes that repeatedly cluster together in phylogenetically remotely related genomes are functionally associated:

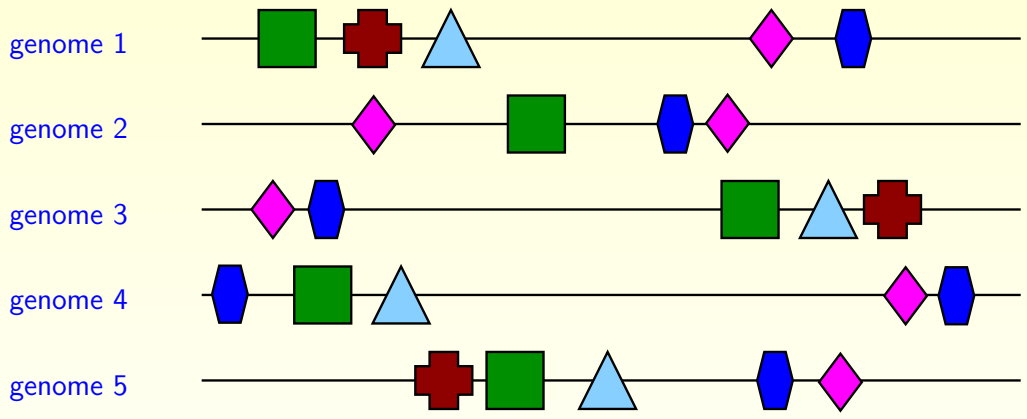
- interacting proteins
- proteins of the same protein complex
- enzymes of the same metabolic pathway

Genome-based gene function prediction

Comparative genomics meets functional genomics.

Idea: Genes that repeatedly cluster together in phylogenetically remotely related genomes are functionally associated:

- interacting proteins
- proteins of the same protein complex
- enzymes of the same metabolic pathway

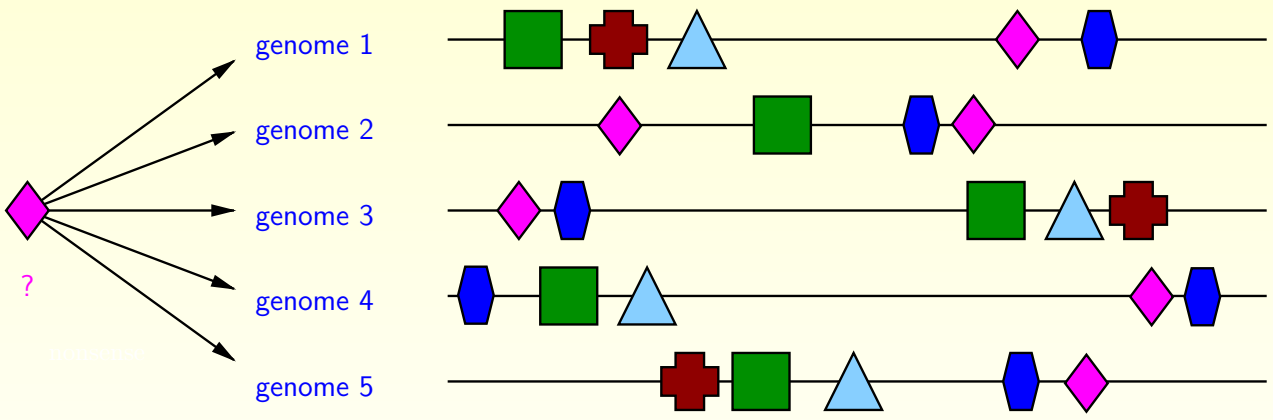


Genome-based gene function prediction

Comparative genomics meets functional genomics.

Idea: Genes that repeatedly cluster together in phylogenetically remotely related genomes are functionally associated:

- interacting proteins
- proteins of the same protein complex
- enzymes of the same metabolic pathway

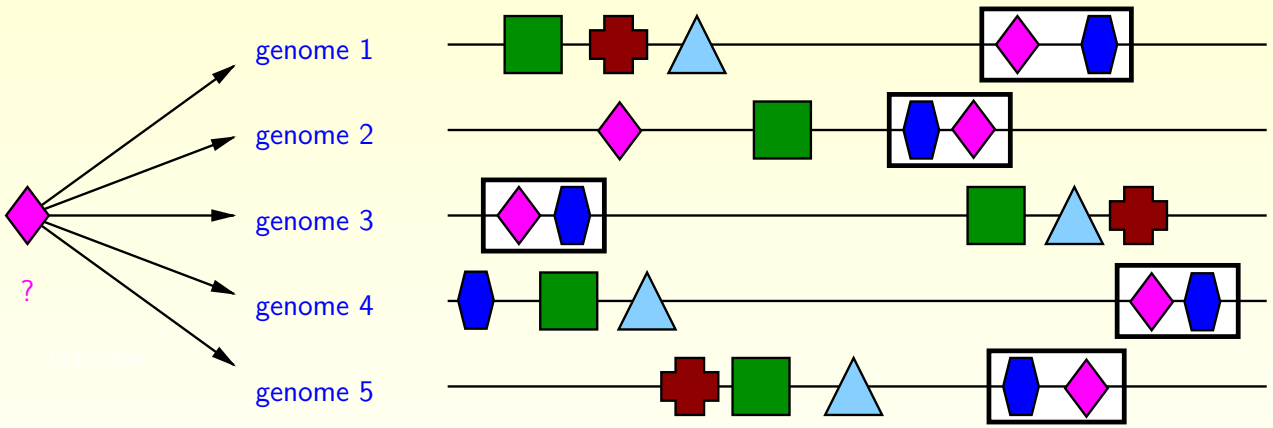


Genome-based gene function prediction

Comparative genomics meets functional genomics.

Idea: Genes that repeatedly cluster together in phylogenetically remotely related genomes are functionally associated:

- interacting proteins
- proteins of the same protein complex
- enzymes of the same metabolic pathway

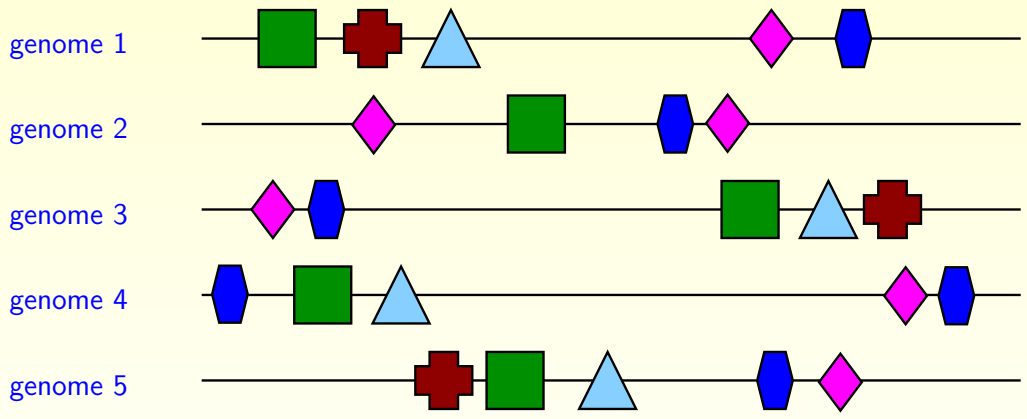


Genome-based gene function prediction

Comparative genomics meets functional genomics.

Idea: Genes that repeatedly cluster together in phylogenetically remotely related genomes are functionally associated:

- interacting proteins
- proteins of the same protein complex
- enzymes of the same metabolic pathway

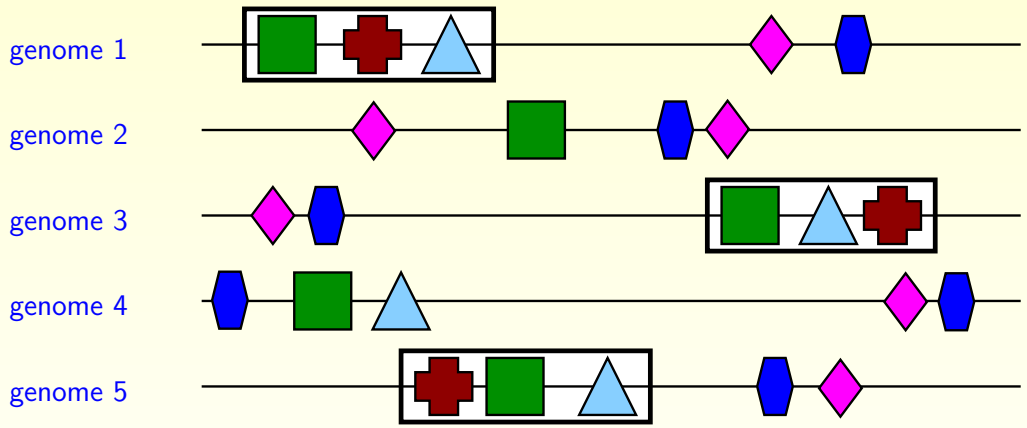


Genome-based gene function prediction

Comparative genomics meets functional genomics.

Idea: Genes that repeatedly cluster together in phylogenetically remotely related genomes are functionally associated:

- interacting proteins
- proteins of the same protein complex
- enzymes of the same metabolic pathway

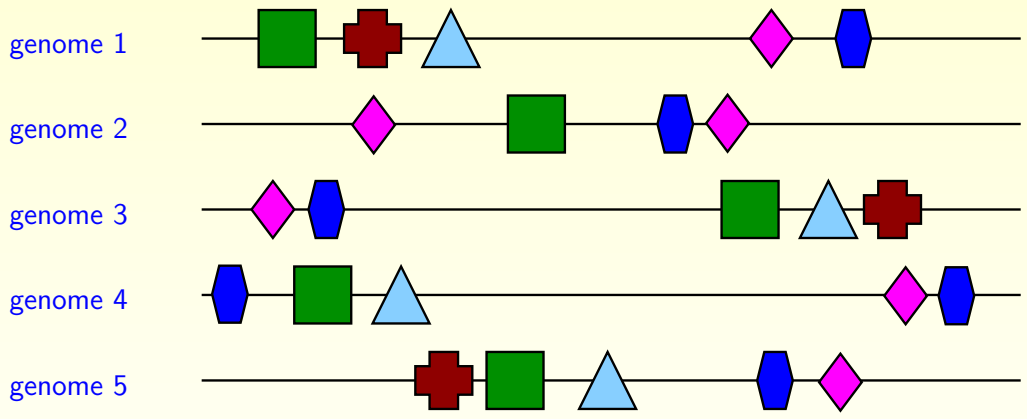


Genome-based gene function prediction

Comparative genomics meets functional genomics.

Idea: Genes that repeatedly cluster together in phylogenetically remotely related genomes are functionally associated:

- interacting proteins
- proteins of the same protein complex
- enzymes of the same metabolic pathway



Genome-based gene function prediction

Comparative genomics meets functional genomics.

Idea: Genes that repeatedly cluster together in phylogenetically remotely related genomes are functionally associated:

- interacting proteins
- proteins of the same protein complex
- enzymes of the same metabolic pathway

Genome-based gene function prediction

Comparative genomics meets functional genomics.

Idea: Genes that repeatedly cluster together in phylogenetically remotely related genomes are functionally associated:

- interacting proteins
- proteins of the same protein complex
- enzymes of the same metabolic pathway

Marcotte *et al.*: Detecting protein function and protein-protein interactions from genome sequences. *Science*, 1999.

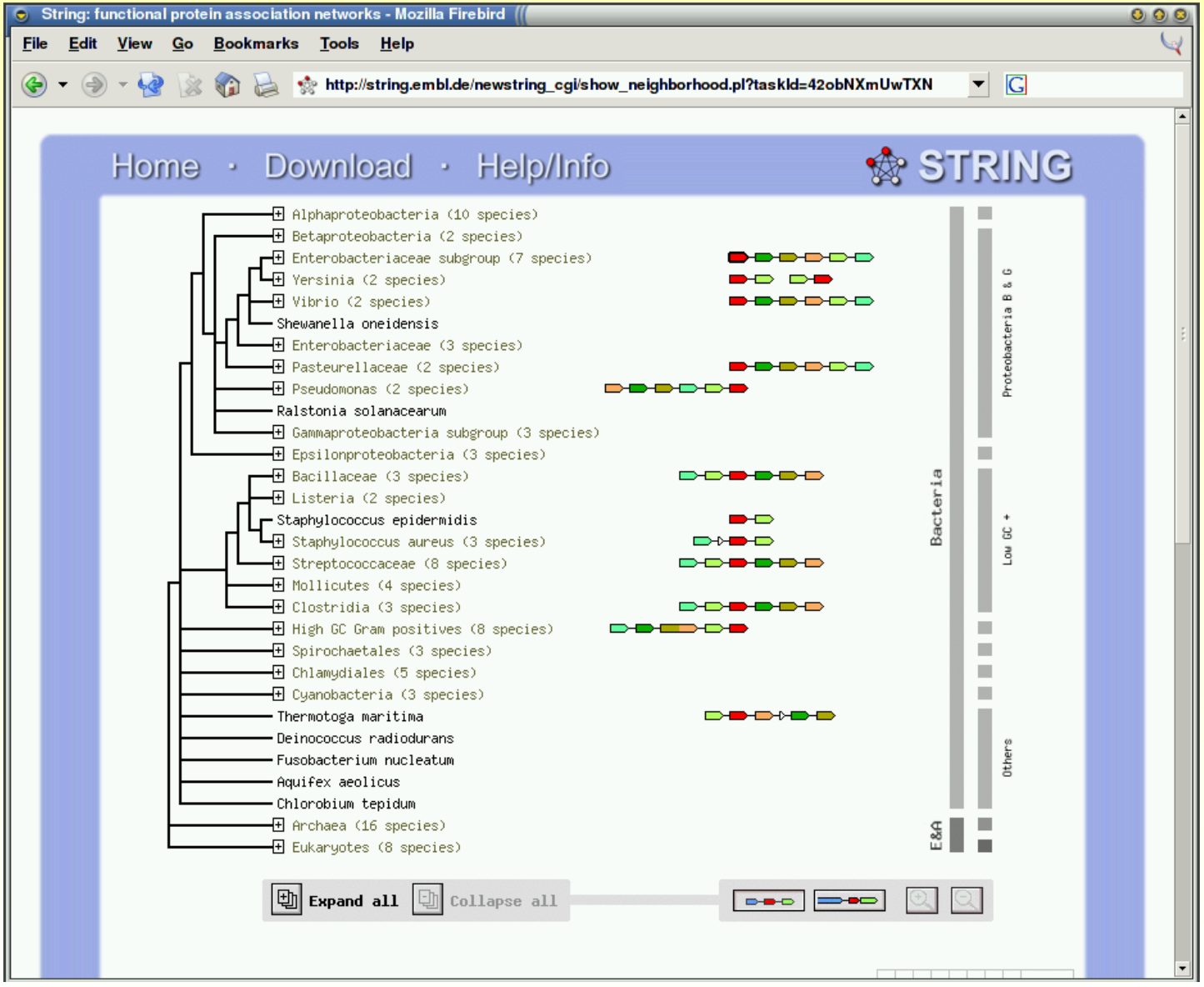
Overbeek *et al.*: The use of gene clusters to infer functional coupling. *PNAS*, 1999.

Snel *et al.*: STRING: A web-server to retrieve and display the repeatedly occurring neighbourhood of a gene, *NAR*, 2000.

Zheng *et al.*: Phylogenetic detection of conserved gene clusters in microbial genomes, *BMC Bioinformatics*, 2005.

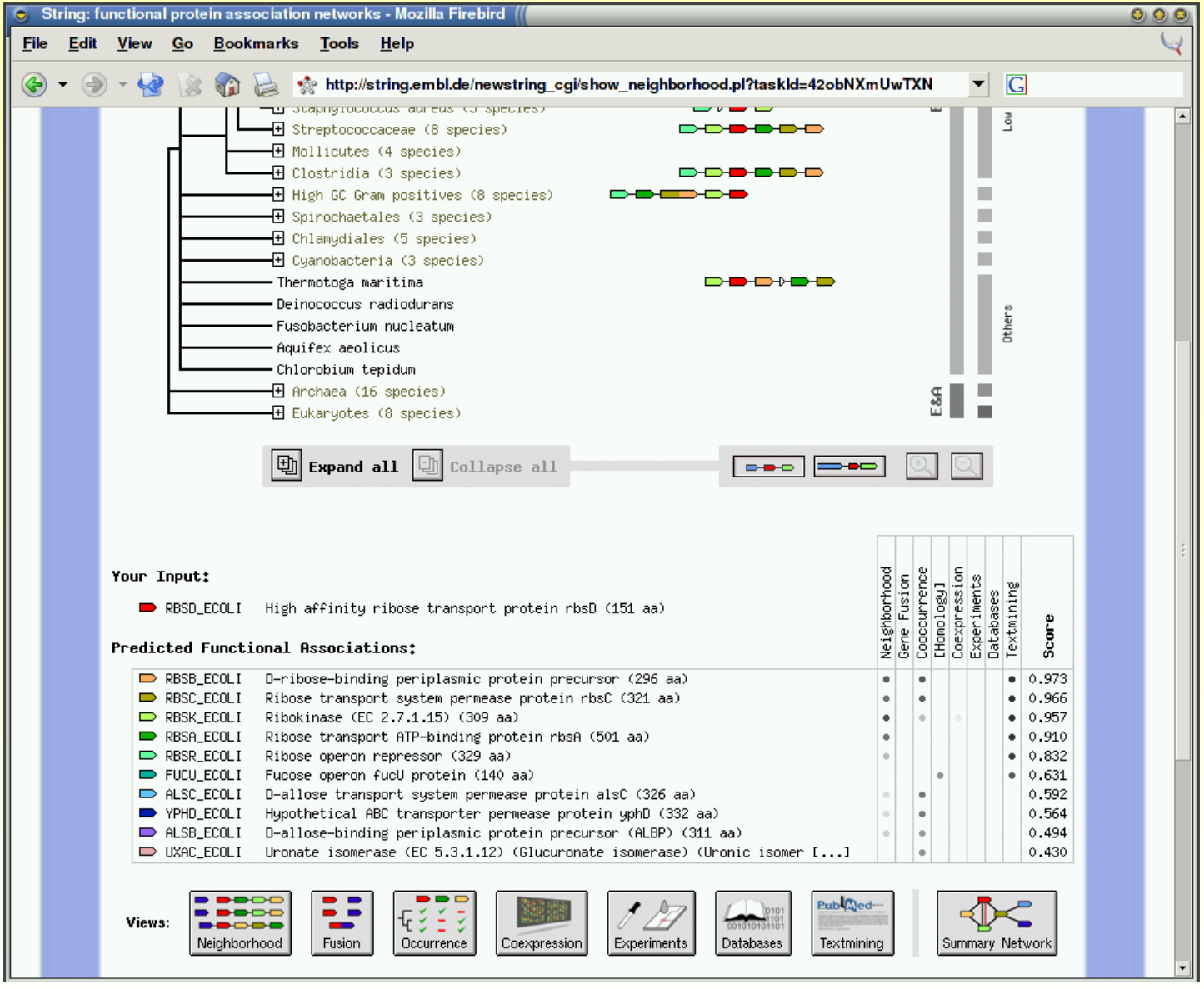
STRING Web server (Snel et al., 2000)

http://string.embl.de/

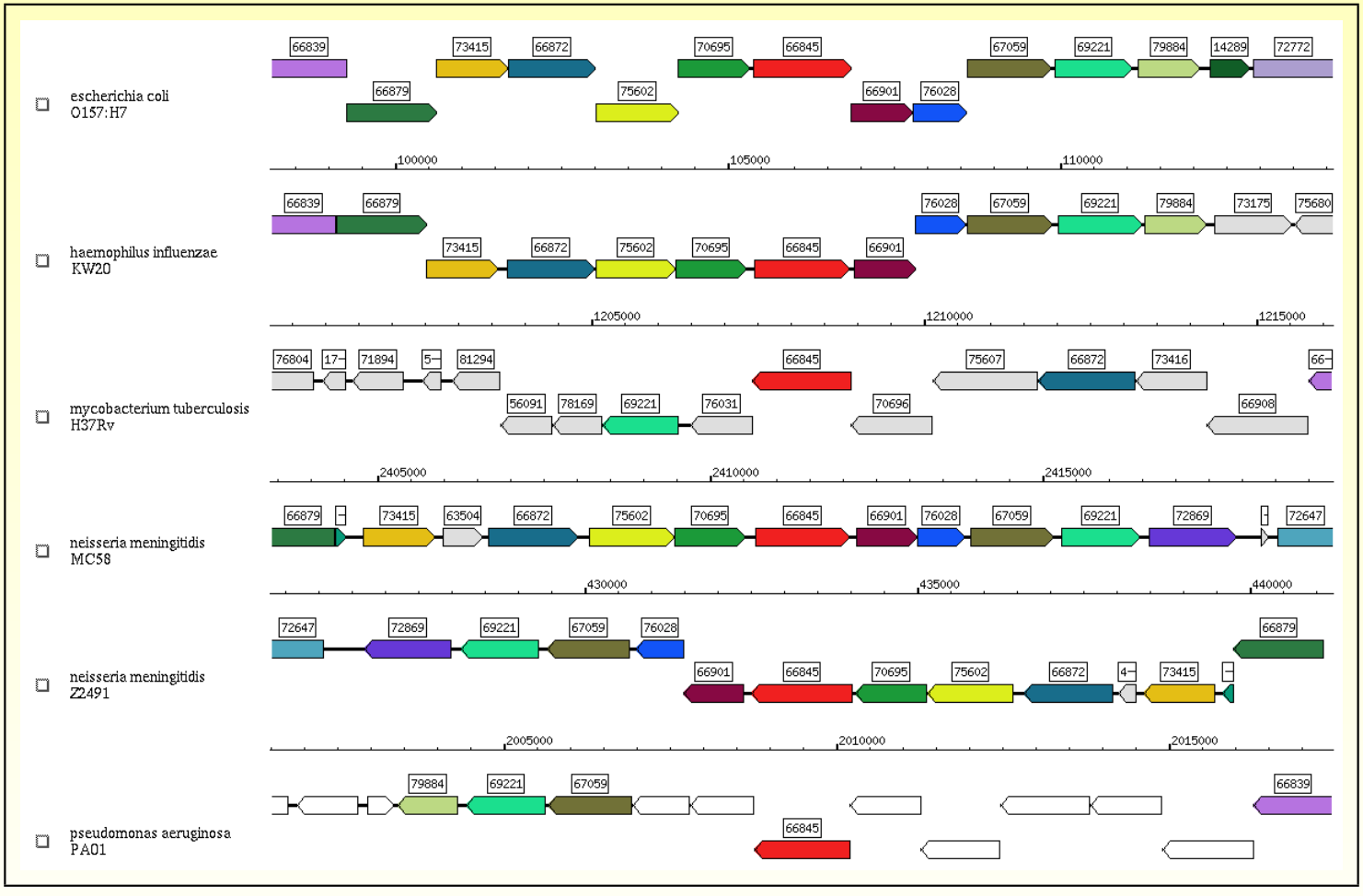


STRING Web server (Snel et al., 2000)

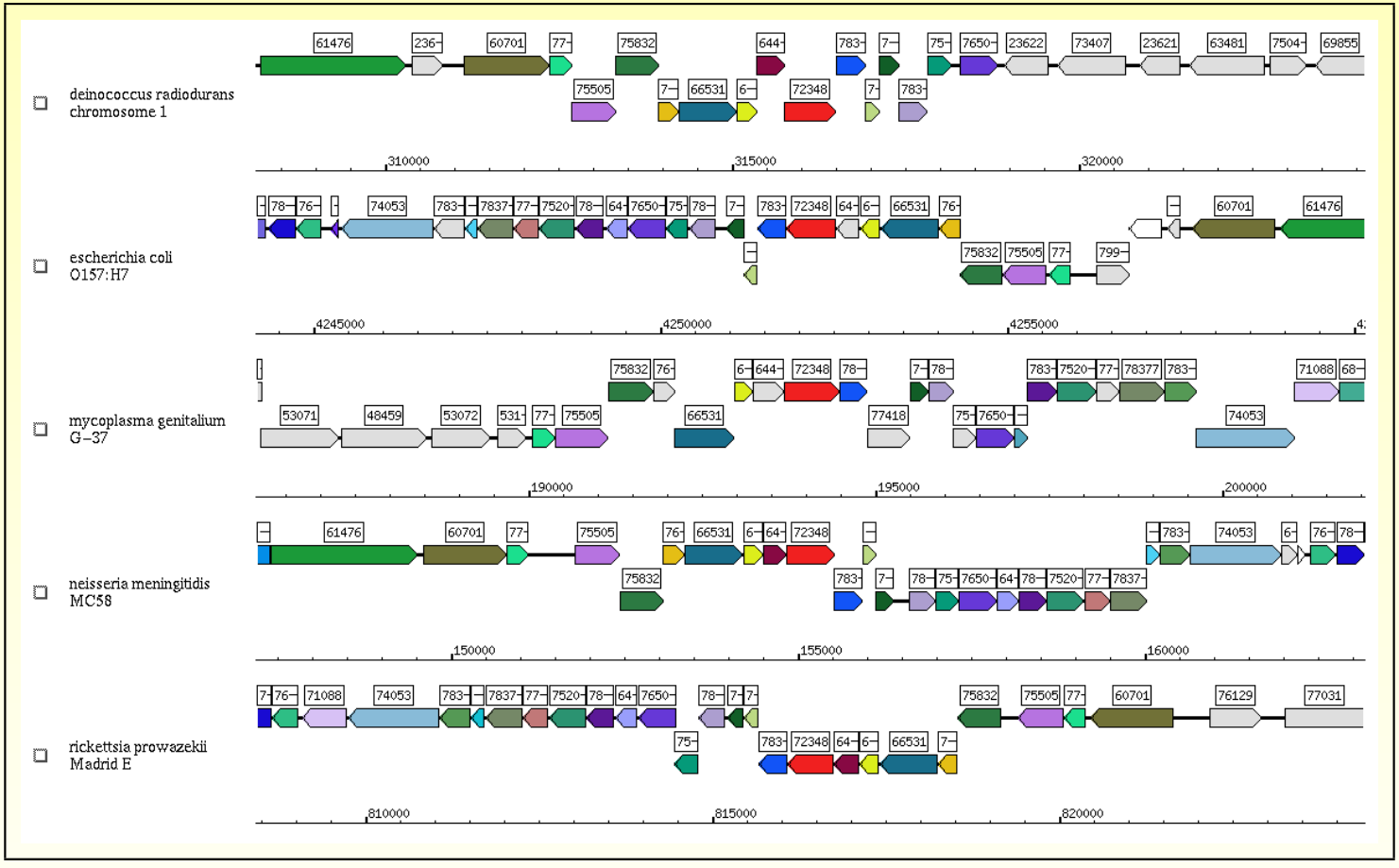
http://string.embl.de/



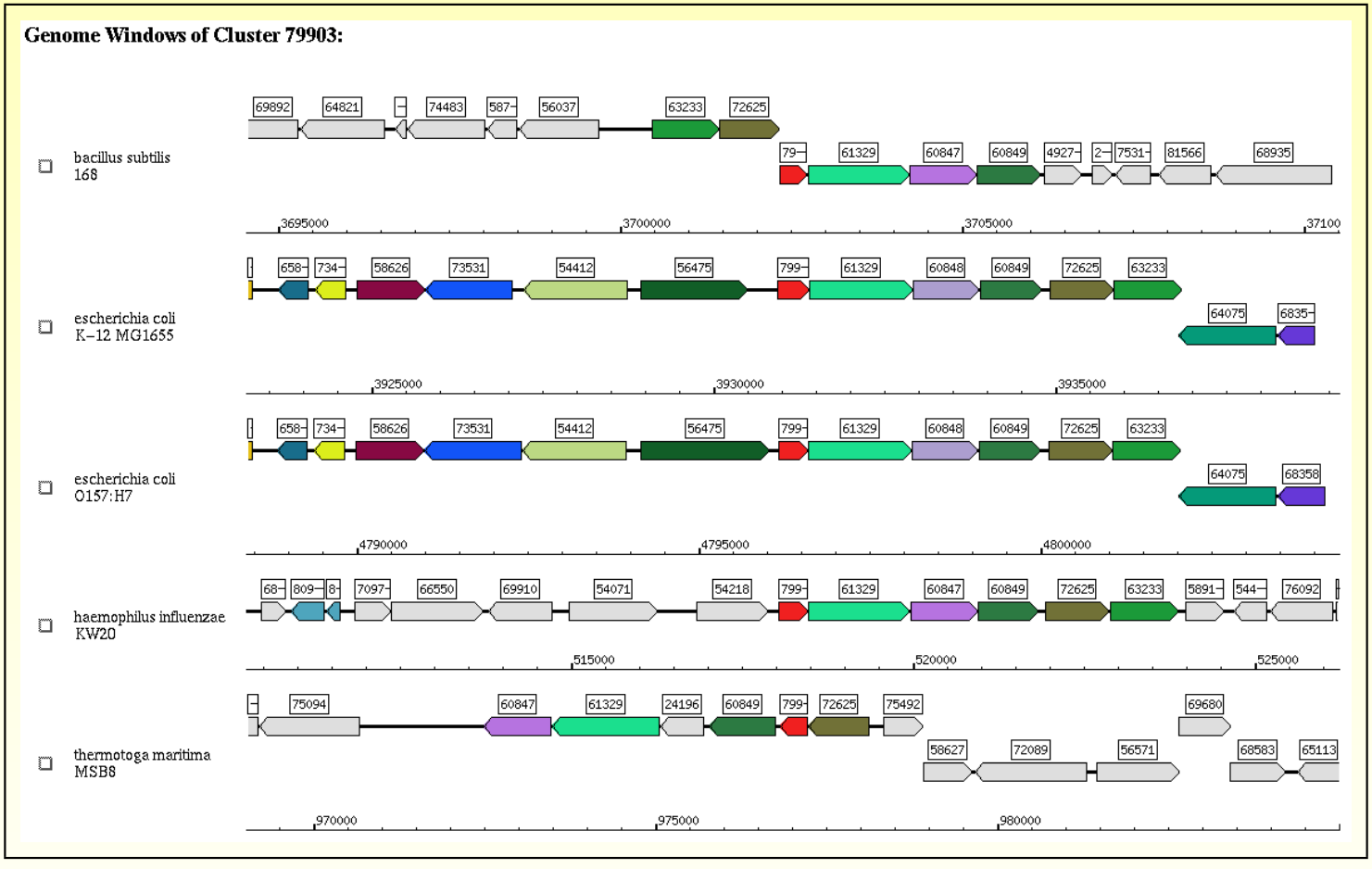
Genome Windows: DCW cluster (division and cell wall)



Genome Windows: Ribosomal Super Operon



Genome Windows: Ribose ABC Transporter



Formalization of gene clusters: common intervals

Given permutations (genomes) $\pi_1, \pi_2, \ldots, \pi_k$ of the numbers (genes) $0, 1, \ldots, n$, find subsets of numbers that occur contiguously in all permutations.

π_1	0	1	2	3	4	5	6	7
π_2	6	7	5	1	4	3	2	0
π_3	7	0	1	2	4	3	5	6

Formalization of gene clusters: common intervals

Given permutations (genomes) $\pi_1, \pi_2, \ldots, \pi_k$ of the numbers (genes) $0, 1, \ldots, n$, find subsets of numbers that occur contiguously in all permutations.

π_1	0	1	2	3	4	5	6	7
π_2	6	7	5	1	4	3	2	0
π_3	7	0	1	2	4	3	5	6

Common intervals: [3,4]

Formalization of gene clusters: common intervals

Given permutations (genomes) $\pi_1, \pi_2, \ldots, \pi_k$ of the numbers (genes) $0, 1, \ldots, n$, find subsets of numbers that occur contiguously in all permutations.

π_1	0	1	2	3	4	5	6	7
π_2	6	7	5	1	4	3	2	0
π_3	7	0	1	2	4	3	5	6

Common intervals: [3,4] [2,4]

Formalization of gene clusters: common intervals

Given permutations (genomes) $\pi_1, \pi_2, \ldots, \pi_k$ of the numbers (genes) $0, 1, \ldots, n$, find subsets of numbers that occur contiguously in all permutations.

π_1	0	1	2	3	4	5	6	7
π_2	6	7	5	1	4	3	2	0
π_3	7	0	1	2	4	3	5	6

Common intervals: [3,4] [2,4] [1,4]

Formalization of gene clusters: common intervals

Given permutations (genomes) $\pi_1, \pi_2, \ldots, \pi_k$ of the numbers (genes) $0, 1, \ldots, n$, find subsets of numbers that occur contiguously in all permutations.

π_1	0	1	2	3	4	5	6	7
π_2	6	7	5	1	4	3	2	0
π_3	7	0	1	2	4	3	5	6

Common intervals: [3,4] [2,4] [1,4] [0,4]

Formalization of gene clusters: common intervals

Given permutations (genomes) $\pi_1, \pi_2, \ldots, \pi_k$ of the numbers (genes) $0, 1, \ldots, n$, find subsets of numbers that occur contiguously in all permutations.

π_1	0	1	2	3	4	5	6	7
π_2	6	7	5	1	4	3	2	0
π_3	7	0	1	2	4	3	5	6

Common intervals: [3,4] [2,4] [1,4] [0,4] [1,5]

Formalization of gene clusters: common intervals

Given permutations (genomes) $\pi_1, \pi_2, \ldots, \pi_k$ of the numbers (genes) $0, 1, \ldots, n$, find subsets of numbers that occur contiguously in all permutations.

π_1	0	1	2	3	4	5	6	7
π_2	6	7	5	1	4	3	2	0
π_3	7	0	1	2	4	3	5	6

Common intervals: [3,4] [2,4] [1,4] [0,4] [1,5] [0,5]

Formalization of gene clusters: common intervals

Given permutations (genomes) $\pi_1, \pi_2, \ldots, \pi_k$ of the numbers (genes) $0, 1, \ldots, n$, find subsets of numbers that occur contiguously in all permutations.

π_1	0	1	2	3	4	5	6	7
π_2	6	7	5	1	4	3	2	0
π_3	7	0	1	2	4	3	5	6

Common intervals: [3,4] [2,4] [1,4] [0,4] [1,5] [0,5] [0,7]

Formalization of gene clusters: common intervals

Given permutations (genomes) $\pi_1, \pi_2, \ldots, \pi_k$ of the numbers (genes) $0, 1, \ldots, n$, find subsets of numbers that occur contiguously in all permutations.

π_1	0	1	2	3	4	5	6	7
π_2	6	7	5	1	4	3	2	0
π_3	7	0	1	2	4	3	5	6

Common intervals: [3,4] [2,4] [1,4] [0,4] [1,5] [0,5] [0,7]

Algorithms:

- Uno & Yagiura, *Algorithmica* 2000:
  Find all common intervals of 2 permutations in $\mathcal{O}(n + |\text{output}|)$ time.
- Heber & JS, *CPM* 2001:
  Find all common intervals of $k \geq 2$ permutations in $\mathcal{O}(kn + |\text{output}|)$ time.

Finding all common intervals of two permutations π_1 and π_2

Let $1 \leq x \leq y \leq n$.

Notation: $\pi([x, y]) := \{\pi(x), \pi(x+1), \ldots, \pi(y)\}$

Definitions:
$$l(x, y) := \min \pi_2([x, y])$$
$$u(x, y) := \max \pi_2([x, y])$$
$$f(x, y) := u(x, y) - l(x, y) - (y - x)$$

Example:

π_1	0	1	2	3	4	5	6	7
	0	1	2	3	4	5	6	7
π_2	6	7	5	1	4	3	2	0
	0	1	2	3	4	5	6	7

Finding all common intervals of two permutations π_1 and π_2

Let $1 \leq x \leq y \leq n$.

Notation: $\pi([x, y]) := \{\pi(x), \pi(x+1), \ldots, \pi(y)\}$

Definitions:
$$l(x, y) := \min \pi_2([x, y])$$
$$u(x, y) := \max \pi_2([x, y])$$
$$f(x, y) := u(x, y) - l(x, y) - (y - x)$$

Example:

π_1	0	1	2	3	4	5	6	7
	0	1	2	3	4	5	6	7
π_2	6	7	5	1	4	3	2	0
	0	1	2	3	4	5	6	7

$f(3, 6) = 4 - 1 - (6 - 3) = 0$

Finding all common intervals of two permutations π_1 and π_2

Let $1 \leq x \leq y \leq n$.

Notation: $\pi([x, y]) := \{\pi(x), \pi(x+1), \ldots, \pi(y)\}$

Definitions:
$$l(x, y) := \min \pi_2([x, y])$$
$$u(x, y) := \max \pi_2([x, y])$$
$$f(x, y) := u(x, y) - l(x, y) - (y - x)$$

Example:

π_1	0	1	2	3	4	5	6	7	$f(3,6) = 4 - 1 - (6 - 3) = 0$
	0	1	2	3	4	5	6	7	
π_2	6	7	5	1	4	3	2	0	$f(1,4) = 7 - 1 - (4 - 1) = 3 > 0$
	0	1	2	3	4	5	6	7	

Finding all common intervals of two permutations π_1 and π_2

Let $1 \leq x \leq y \leq n$.

Notation: $\pi([x, y]) := \{\pi(x), \pi(x+1), \ldots, \pi(y)\}$

Definitions:
$$l(x, y) := \min \pi_2([x, y])$$
$$u(x, y) := \max \pi_2([x, y])$$
$$f(x, y) := u(x, y) - l(x, y) - (y - x)$$

Example:

π_1	0	1	2	3	4	5	6	7
	0	1	2	3	4	5	6	7
π_2	6	7	5	1	4	3	2	0
	0	1	2	3	4	5	6	7

$f(3, 6) = 4 - 1 - (6 - 3) = 0$

$f(1, 4) = 7 - 1 - (4 - 1) = 3 > 0$

Simple algorithm: For all $1 \leq x \leq y \leq n$ test if $f(x, y) = 0$.

Finding all common intervals of two permutations π_1 and π_2

Let $1 \leq x \leq y \leq n$.

Notation: $\pi([x, y]) := \{\pi(x), \pi(x+1), \ldots, \pi(y)\}$

Definitions:
$$l(x, y) := \min \pi_2([x, y])$$
$$u(x, y) := \max \pi_2([x, y])$$
$$f(x, y) := u(x, y) - l(x, y) - (y - x)$$

Example:

π_1	0	1	2	3	4	5	6	7	$f(3,6) = 4 - 1 - (6 - 3) = 0$
	0	1	2	3	4	5	6	7	
π_2	6	7	5	1	4	3	2	0	$f(1,4) = 7 - 1 - (4 - 1) = 3 > 0$
	0	1	2	3	4	5	6	7	

Simple algorithm: For all $1 \leq x \leq y \leq n$ test if $f(x, y) = 0$.

Analysis: $\mathcal{O}(n^2)$ time.

Finding all common intervals of two permutations π_1 and π_2

Uno & Yagiura, 2000:

Perform the test $f(x, y) = 0$ not for all pairs (x, y).

Definition:

For given x, call a value of $y > x$ *wasteful*, if and only if for all $x' \leq x$:

$$f(x', y) > 0.$$

Lemma:

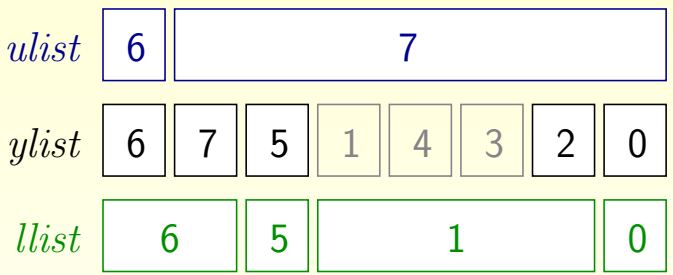
For fixed x, $f(x, y)$ increases monotonically for the non-wasteful indices y $(> x)$.

Algorithm (Idea):

- x runs in right-to-left direction through a doubly linked list *ylist* that initially contains the entries of π_2.
- In each step, the entries of wasteful indices y $(> x)$ are removed.
- Test for the remaining $y > x$ in *ylist* from left to right if $f(x, y) = 0$.

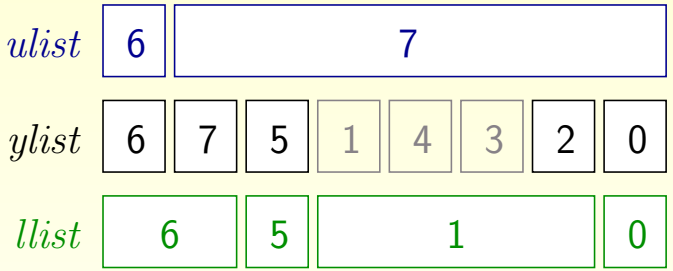
Algorithm RC (Uno & Yagiura)

- Removal of wasteful indices from *ylist* is done by means of two additional lists *llist* and *ulist* that implement the functions l and u.
- The elements of *llist* and *ulist* are maximal intervals of *ylist* with the same smallest resp. largest element.



Algorithm RC (Uno & Yagiura)

- Removal of wasteful indices from *ylist* is done by means of two additional lists *llist* and *ulist* that implement the functions l and u.
- The elements of *llist* and *ulist* are maximal intervals of *ylist* with the same smallest resp. largest element.



Analysis:
$\mathcal{O}(n + |\text{output}|)$ time, $\mathcal{O}(n)$ space.

Finding all common intervals of $k \geq 2$ permutations

Obvious generalization:

Given k permutations $\pi_1, \pi_2, \ldots, \pi_k$.

For $j = 2, 3, \ldots, k$ compute the common intervals of π_1 and π_j.

Output all intervals that are found in all of these comparisons.

π_1	0	1	2	3	4	5	6	7
π_2	6	7	5	1	4	3	2	0
π_3	7	0	1	2	4	3	5	6

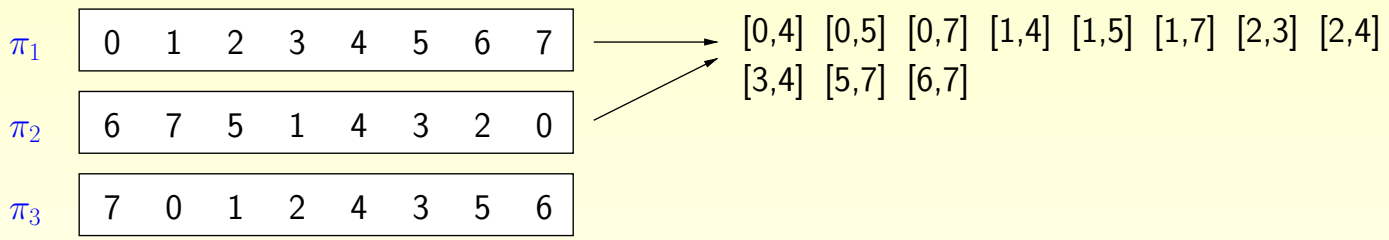
Finding all common intervals of $k \geq 2$ permutations

Obvious generalization:

Given k permutations $\pi_1, \pi_2, \ldots, \pi_k$.

For $j = 2, 3, \ldots, k$ compute the common intervals of π_1 and π_j.

Output all intervals that are found in all of these comparisons.



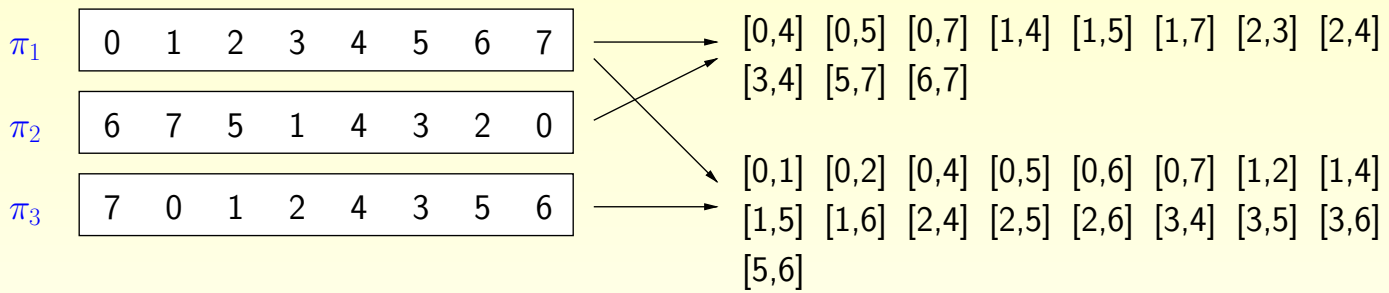
Finding all common intervals of $k \geq 2$ permutations

Obvious generalization:

Given k permutations $\pi_1, \pi_2, \ldots, \pi_k$.

For $j = 2, 3, \ldots, k$ compute the common intervals of π_1 and π_j.

Output all intervals that are found in all of these comparisons.



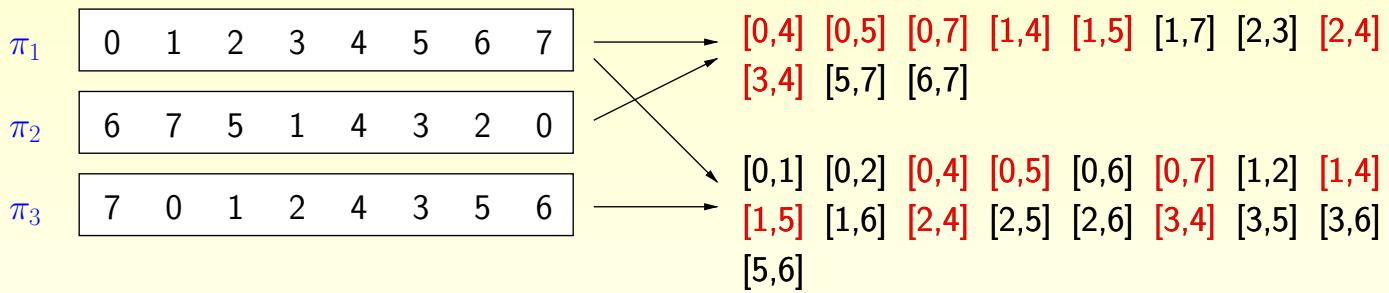
Finding all common intervals of $k \geq 2$ permutations

Obvious generalization:

Given k permutations $\pi_1, \pi_2, \ldots, \pi_k$.

For $j = 2, 3, \ldots, k$ compute the common intervals of π_1 and π_j.

Output all intervals that are found in all of these comparisons.



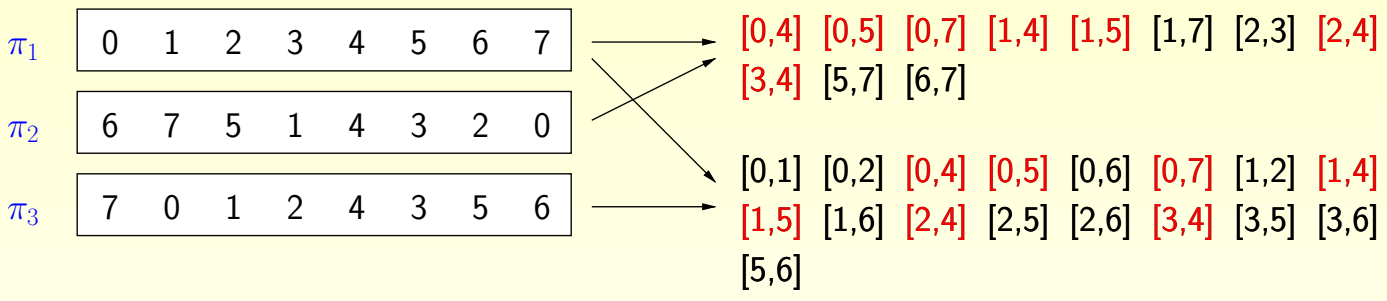
Finding all common intervals of $k \geq 2$ permutations

Obvious generalization:

Given k permutations $\pi_1, \pi_2, \ldots, \pi_k$.

For $j = 2, 3, \ldots, k$ compute the common intervals of π_1 and π_j.

Output all intervals that are found in all of these comparisons.



Analysis:

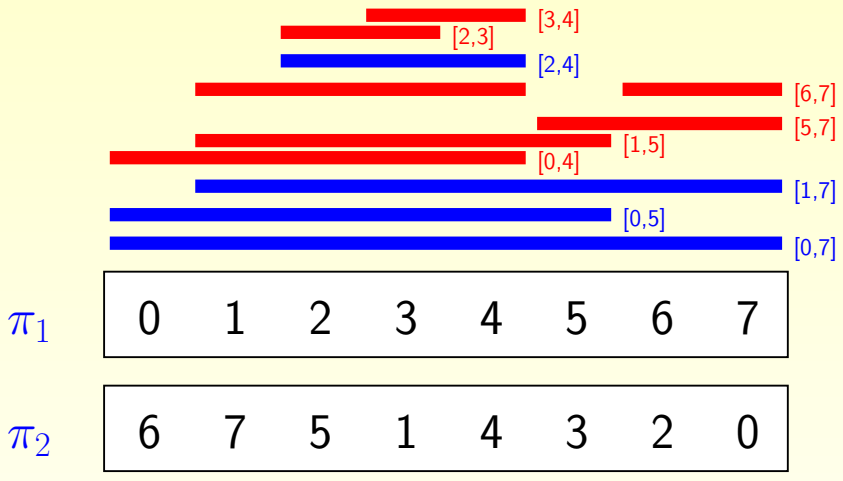
$\mathcal{O}(kn + \sum |K_i|)$ time

where K_i = the number of common intervals of π_1 and π_i.

Irreducible Intervals

Goal: An algorithm with output-dependent time complexity $\mathcal{O}(kn + |\text{output}|)$.

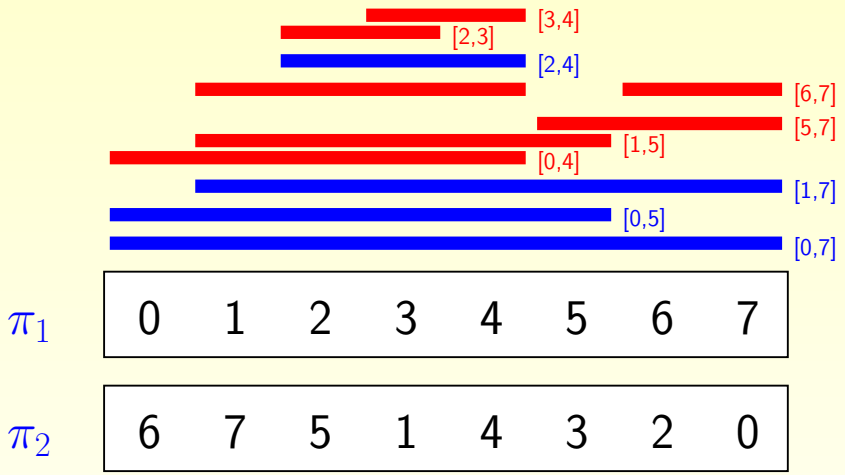
Observation: Common intervals form *"chains"* of non-trivially overlapping intervals.



Irreducible Intervals

Goal: An algorithm with output-dependent time complexity $\mathcal{O}(kn + |\text{output}|)$.

Observation: Common intervals form *"chains"* of non-trivially overlapping intervals.



Definition:

A common interval c is reducible if there exists a non-trivial chain that generates c, otherwise it is irreducible.

Properties of irreducible intervals

Lemma:

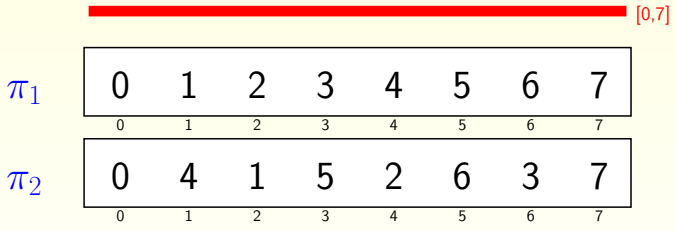
The subchains of all the maximal chains of irreducible intervals generate exactly all common intervals.

Theorem:

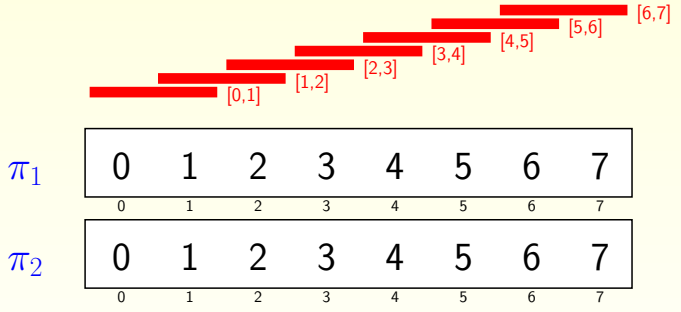
For the number of irreducible intervals K the following holds:

$$1 \leq K \leq n - 1$$

Example:



$K = 1$

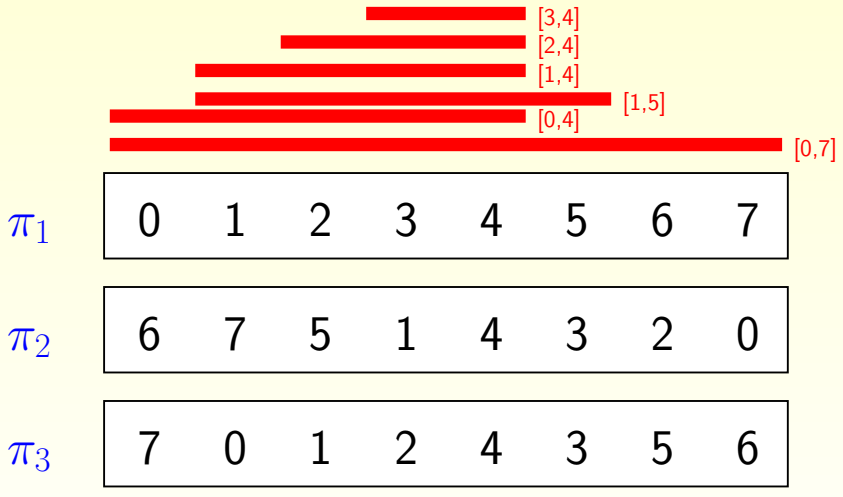


$K = n - 1$

Finding all common intervals of k $\geq$ 2 permutations

Algorithm:

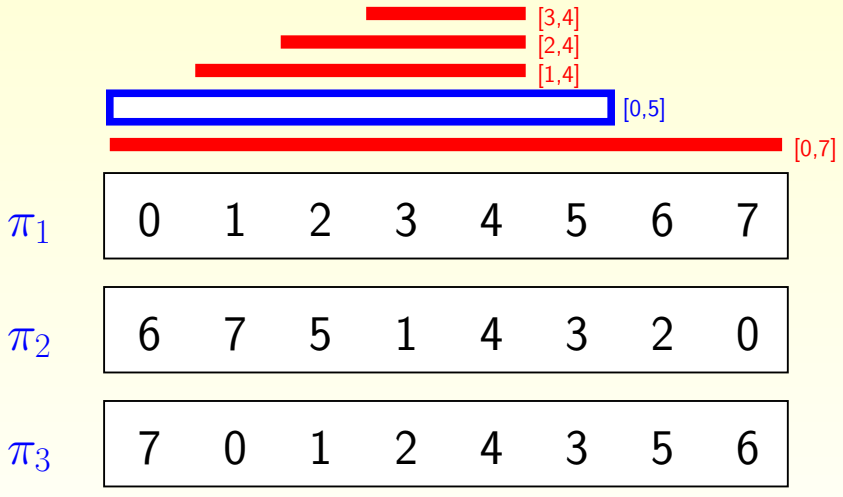
- Find the set of all irreducible intervals.
- Partition this set into maximal chains of non-trivially overlapping intervals.
- For each such chain generate all subchains: the common intervals.



Finding all common intervals of k $\geq$ 2 permutations

Algorithm:

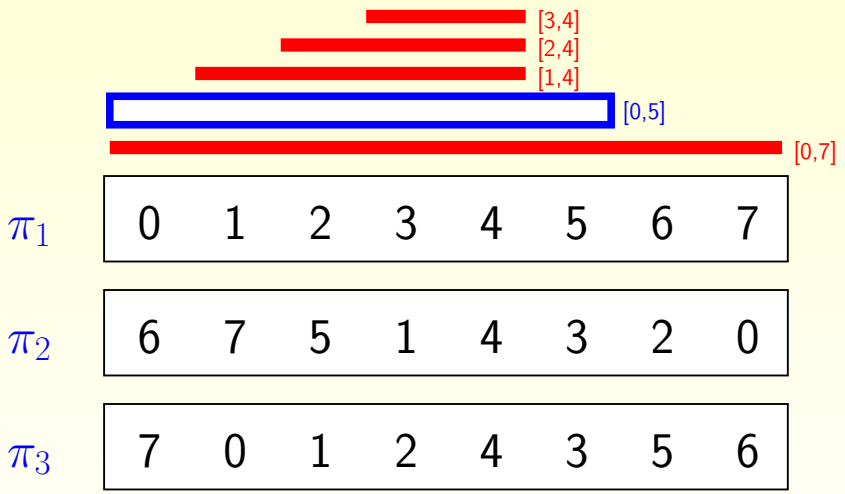
- Find the set of all irreducible intervals.
- Partition this set into maximal chains of non-trivially overlapping intervals.
- For each such chain generate all subchains: the common intervals.



Finding all common intervals of k $\geq$ 2 permutations

Algorithm:

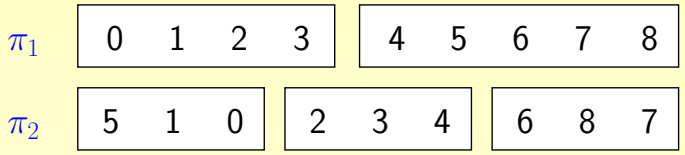
- Find the set of all irreducible intervals.
- Partition this set into maximal chains of non-trivially overlapping intervals.
- For each such chain generate all subchains: the common intervals.



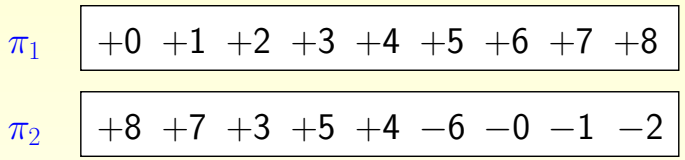
Analysis: $\mathcal{O}(kn + |\text{output}|)$ time, $\mathcal{O}(n)$ additional space

More realistic genome models

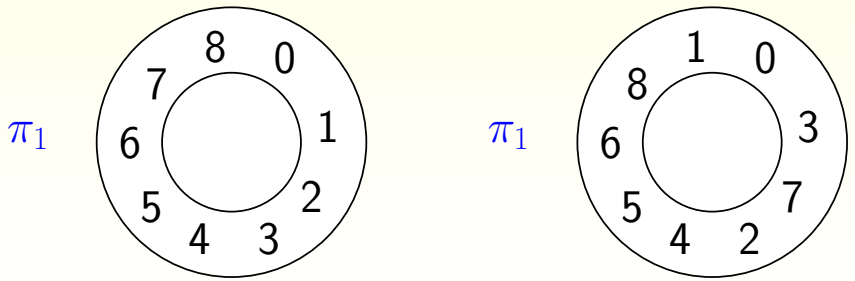
1. Genomes of higher organisms often have more than one chromosome
   $\Rightarrow$ *multichromosomal permutations*



2. Genes of a cluster should lie on the same DNA strand
   $\Rightarrow$ *signed permutations*



3. Bacterial, archaeal, and mitochondrial DNA is often circular
   $\Rightarrow$ *circular permutations*



Further extensions of the model

Generalization

- Allow paralogous genes (gene families)

Relaxation:

- Common intervals do not have to be contained in all genomes
- Not each gene has always to occur

Other applications

- Inverse of *Consective Arrangement Problem*
- Candidates for *Subtour Exchange Crossover* in genetic algorithms
- Model-free distance for phylogenetic analyses

Inclusion of paralogous genes

Problem:
In case of *duplicated genes*, it is difficult to assign correct orthologous gene pairs.
Possibly *the* ortholog does not even exist.

Consequence:
Do not distinguish between paralogous gene copies.

New model:
Use the same element (number) more than once for paralogous copies of genes.
$\longrightarrow$ genomes are modeled as *sequences* instead of permutations.

Formal model

Given: k sequences $\mathcal{S} = (S_1, S_2, \ldots, S_k)$ over an alphabet Σ.

Common interval:
a subset $C \subseteq \Sigma$ whose elements occur contiguously in each $S_l \in \mathcal{S}$.

Goal:
Find all maximal occurrences of common intervals in $\mathcal{S}$.

Example:

S_1	3	1	2	3	1	5	2	6
S_2	4	3	5	5	5	1	4	2	2
S_3	7	5	1	5	3	6	5	

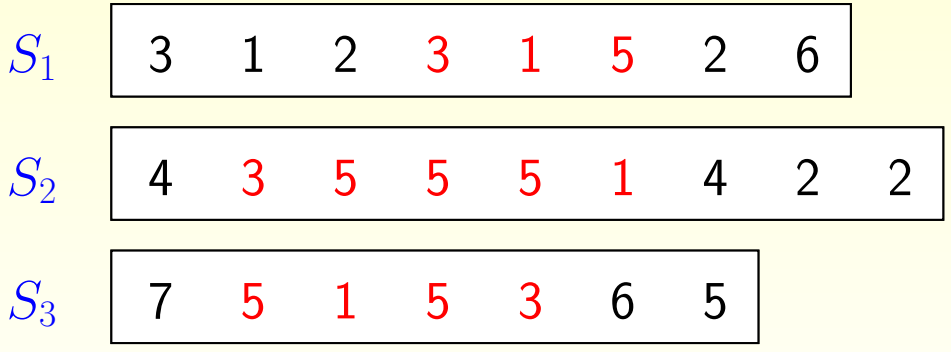
Formal model

Given: k sequences $\mathcal{S} = (S_1, S_2, \ldots, S_k)$ over an alphabet Σ.

Common interval:
a subset $C \subseteq \Sigma$ whose elements occur contiguously in each $S_l \in \mathcal{S}$.

Goal:
Find all maximal occurrences of common intervals in $\mathcal{S}$.

Example:



Common intervals: {1,3,5}

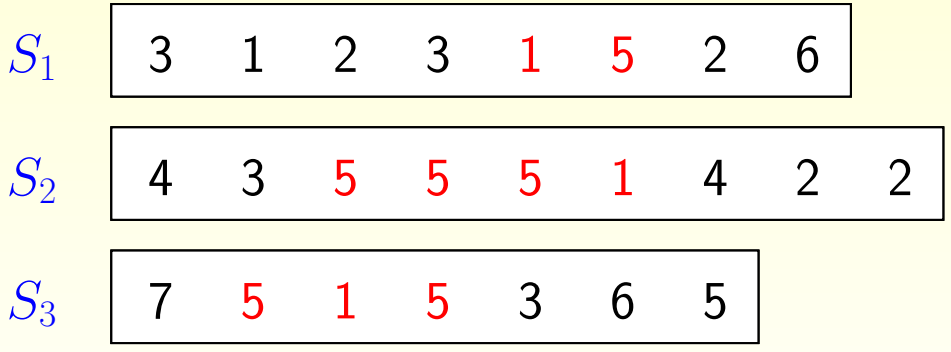
Formal model

Given: k sequences $\mathcal{S} = (S_1, S_2, \ldots, S_k)$ over an alphabet Σ.

Common interval:
a subset $C \subseteq \Sigma$ whose elements occur contiguously in each $S_l \in \mathcal{S}$.

Goal:
Find all maximal occurrences of common intervals in $\mathcal{S}$.

Example:



Common intervals: {1,3,5} {1,5}

Formal model

Given: k sequences $\mathcal{S} = (S_1, S_2, \ldots, S_k)$ over an alphabet Σ.

Common interval:
a subset $C \subseteq \Sigma$ whose elements occur contiguously in each $S_l \in \mathcal{S}$.

Goal:
Find all maximal occurrences of common intervals in $\mathcal{S}$.

Example:

S_1	3	1	2	3	1	5	2	6
S_2	4	3	5	5	5	1	4	2	2
S_3	7	5	1	5	3	6	5	

Common intervals: {1,3,5} {1,5} {5}

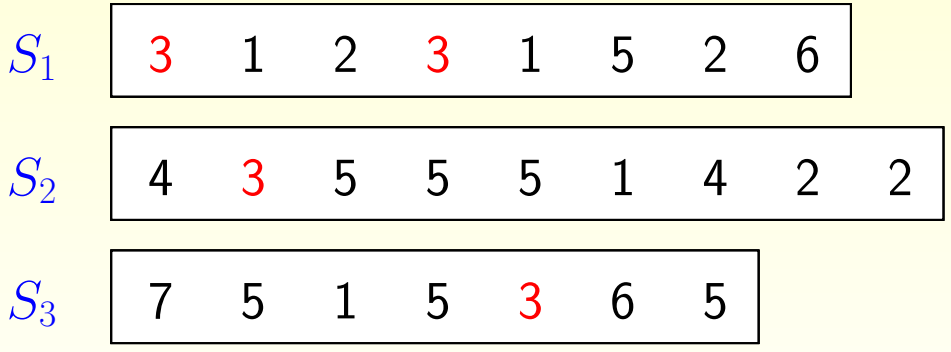
Formal model

Given: k sequences $\mathcal{S} = (S_1, S_2, \ldots, S_k)$ over an alphabet Σ.

Common interval:
a subset $C \subseteq \Sigma$ whose elements occur contiguously in each $S_l \in \mathcal{S}$.

Goal:
Find all maximal occurrences of common intervals in $\mathcal{S}$.

Example:



Common intervals: {1,3,5} {1,5} {5} {3}

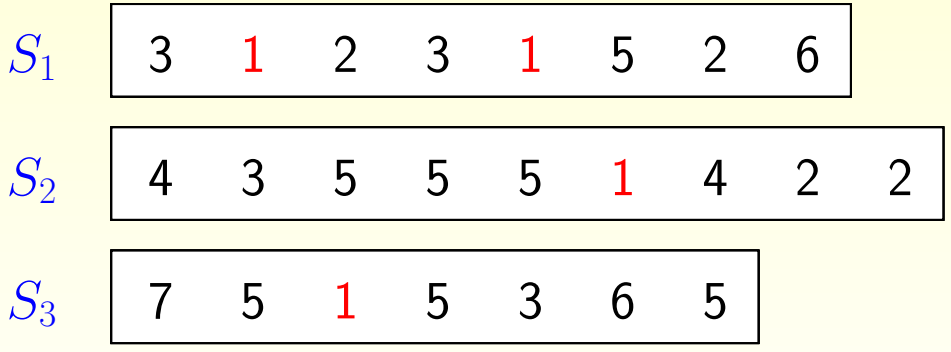
Formal model

Given: k sequences $\mathcal{S} = (S_1, S_2, \ldots, S_k)$ over an alphabet Σ.

Common interval:
a subset $C \subseteq \Sigma$ whose elements occur contiguously in each $S_l \in \mathcal{S}$.

Goal:
Find all maximal occurrences of common intervals in $\mathcal{S}$.

Example:



Common intervals: {1,3,5} {1,5} {5} {3} {1}

Formal model

Given: k sequences $\mathcal{S} = (S_1, S_2, \ldots, S_k)$ over an alphabet Σ.

Common interval:
a subset $C \subseteq \Sigma$ whose elements occur contiguously in each $S_l \in \mathcal{S}$.

Goal:
Find all maximal occurrences of common intervals in $\mathcal{S}$.

Example:

S_1	3	1	2	3	1	5	2	6
S_2	4	3	5	5	5	1	4	2	2
S_3	7	5	1	5	3	6	5	

Common intervals: {1,3,5} {1,5} {5} {3} {1}

An elementary algorithm for two sequences

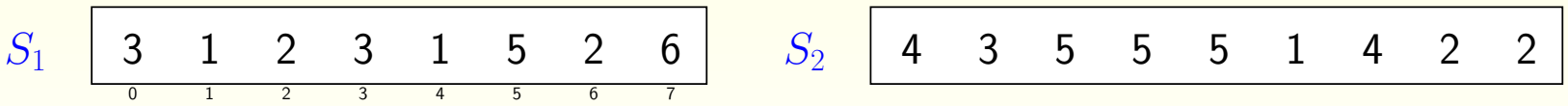
Preprocessing: compute two tables for $S_1 = (3, 1, 2, 3, 1, 5, 2, 6)$:

$POS[1] = 2, 5$
$POS[2] = 3, 7$
$POS[3] = 1, 4$
$POS[4] = \textit{empty}$
$POS[5] = 6$
$POS[6] = 8$

$NUM(i, j)$:

$i \backslash j$	0	1	2	3	4	5	6	7
0	1	2	3	3	3	4	4	5
1		1	2	3	3	4	4	5
2			1	2	3	4	4	5
3				1	2	3	4	5
4					1	2	3	4
5						1	2	3
6							1	2
7								1

Connecting Intervals Algorithm (Schmidt & JS, *CPM 2004*):
While reading S_2, mark in S_1 the observed characters and track maximal intervals of marked characters.



An elementary algorithm for two sequences

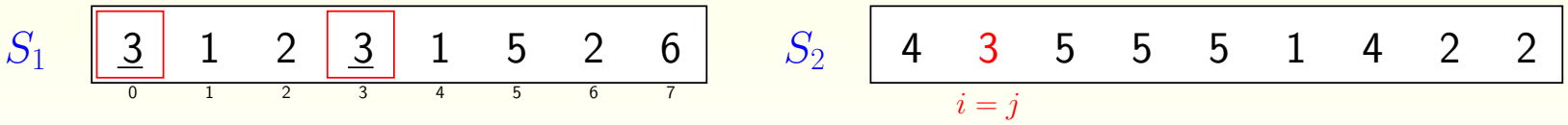
Preprocessing: compute two tables for $S_1 = (3, 1, 2, 3, 1, 5, 2, 6)$:

$POS[1] = 2, 5$
$POS[2] = 3, 7$
$POS[3] = 1, 4$
$POS[4] = \textit{empty}$
$POS[5] = 6$
$POS[6] = 8$

$NUM(i, j)$:

$i \backslash j$	0	1	2	3	4	5	6	7
0	1	2	3	3	3	4	4	5
1		1	2	3	3	4	4	5
2			1	2	3	4	4	5
3				1	2	3	4	5
4					1	2	3	4
5						1	2	3
6							1	2
7								1

Connecting Intervals Algorithm (Schmidt & JS, *CPM 2004*):
While reading S_2, mark in S_1 the observed characters and track maximal intervals of marked characters.



An elementary algorithm for two sequences

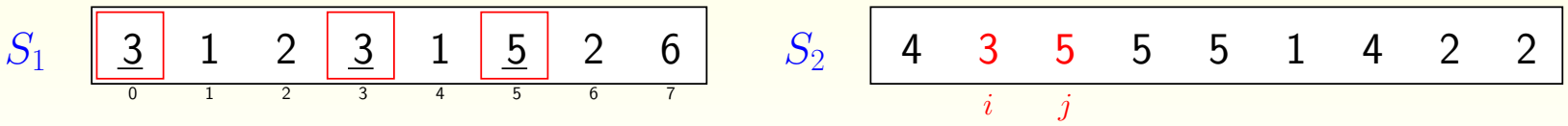
Preprocessing: compute two tables for $S_1 = (3, 1, 2, 3, 1, 5, 2, 6)$:

$POS[1] = 2, 5$
$POS[2] = 3, 7$
$POS[3] = 1, 4$
$POS[4] = \textit{empty}$
$POS[5] = 6$
$POS[6] = 8$

$NUM(i, j)$:

$i \backslash j$	0	1	2	3	4	5	6	7
0	1	2	3	3	3	4	4	5
1		1	2	3	3	4	4	5
2			1	2	3	4	4	5
3				1	2	3	4	5
4					1	2	3	4
5						1	2	3
6							1	2
7								1

Connecting Intervals Algorithm (Schmidt & JS, *CPM 2004*):
While reading S_2, mark in S_1 the observed characters and track maximal intervals of marked characters.



An elementary algorithm for two sequences

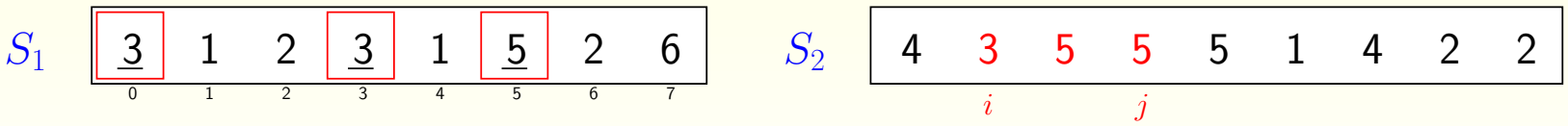
Preprocessing: compute two tables for $S_1 = (3, 1, 2, 3, 1, 5, 2, 6)$:

$POS[1] = 2, 5$
$POS[2] = 3, 7$
$POS[3] = 1, 4$
$POS[4] = \textit{empty}$
$POS[5] = 6$
$POS[6] = 8$

$NUM(i, j)$:

$i \backslash j$	0	1	2	3	4	5	6	7
0	1	2	3	3	3	4	4	5
1		1	2	3	3	4	4	5
2			1	2	3	4	4	5
3				1	2	3	4	5
4					1	2	3	4
5						1	2	3
6							1	2
7								1

Connecting Intervals Algorithm (Schmidt & JS, *CPM 2004*):
While reading S_2, mark in S_1 the observed characters and track maximal intervals of marked characters.



An elementary algorithm for two sequences

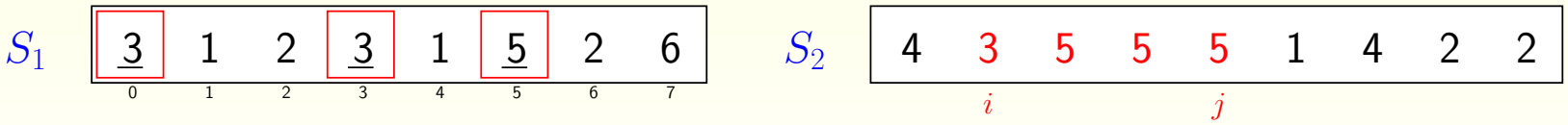
Preprocessing: compute two tables for $S_1 = (3, 1, 2, 3, 1, 5, 2, 6)$:

$POS[1] = 2, 5$
$POS[2] = 3, 7$
$POS[3] = 1, 4$
$POS[4] = \textit{empty}$
$POS[5] = 6$
$POS[6] = 8$

$NUM(i, j)$:

$i \backslash j$	0	1	2	3	4	5	6	7
0	1	2	3	3	3	4	4	5
1		1	2	3	3	4	4	5
2			1	2	3	4	4	5
3				1	2	3	4	5
4					1	2	3	4
5						1	2	3
6							1	2
7								1

Connecting Intervals Algorithm (Schmidt & JS, *CPM 2004*):
While reading S_2, mark in S_1 the observed characters and track maximal intervals of marked characters.



An elementary algorithm for two sequences

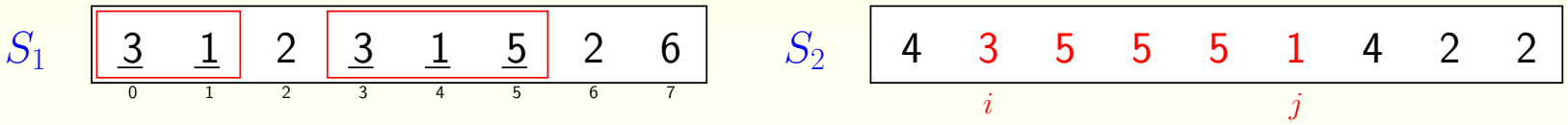
Preprocessing: compute two tables for $S_1 = (3, 1, 2, 3, 1, 5, 2, 6)$:

$POS[1] = 2, 5$
$POS[2] = 3, 7$
$POS[3] = 1, 4$
$POS[4] = \textit{empty}$
$POS[5] = 6$
$POS[6] = 8$

$NUM(i, j)$:

$i \backslash j$	0	1	2	3	4	5	6	7
0	1	2	3	3	3	4	4	5
1		1	2	3	3	4	4	5
2			1	2	3	4	4	5
3				1	2	3	4	5
4					1	2	3	4
5						1	2	3
6							1	2
7								1

Connecting Intervals Algorithm (Schmidt & JS, *CPM 2004*):
While reading S_2, mark in S_1 the observed characters and track maximal intervals of marked characters.



An elementary algorithm for two sequences

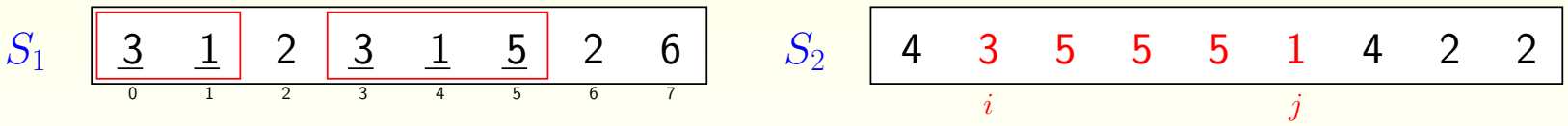
Preprocessing: compute two tables for $S_1 = (3, 1, 2, 3, 1, 5, 2, 6)$:

$POS[1] = 2, 5$
$POS[2] = 3, 7$
$POS[3] = 1, 4$
$POS[4] = \textit{empty}$
$POS[5] = 6$
$POS[6] = 8$

$NUM(i, j)$:

$i \backslash j$	0	1	2	3	4	5	6	7
0	1	2	3	3	3	4	4	5
1		1	2	3	3	4	4	5
2			1	2	3	4	4	5
3				1	2	3	4	5
4					1	2	3	4
5						1	2	3
6							1	2
7								1

Connecting Intervals Algorithm (Schmidt & JS, *CPM 2004*):
While reading S_2, mark in S_1 the observed characters and track maximal intervals of marked characters.



Analysis: $\mathcal{O}(n^2)$ time and space.

More algorithms

Space reduction:

- A different algorithm based on work by Didier (*WABI*, 2003) finds all common intervals of two sequences in $\mathcal{O}(n^2)$ time and $\mathcal{O}(n)$ space.

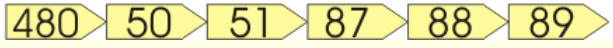
More than two sequences:

- Find all common intervals in k sequences in $\mathcal{O}(kn^2)$ time and space.
- Find all common intervals that appear in at least k' out of k given sequences in $\mathcal{O}(k(1+k-k')n^2)$ time and $\mathcal{O}(kn^2)$ space.

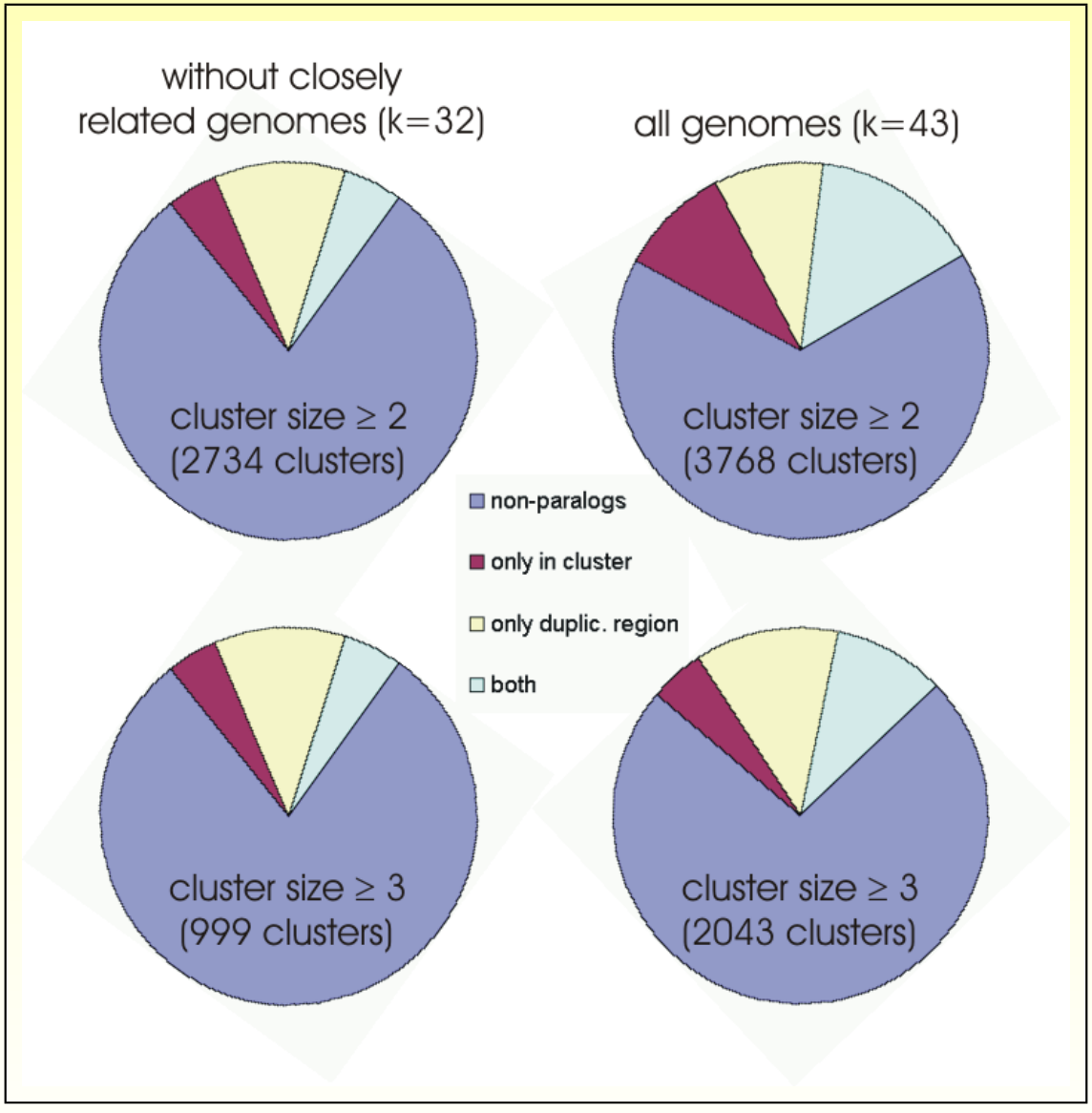
Experimental results. Data source: COG

Aquifex aeolicus complete genome - 0..1551335
1529 proteins

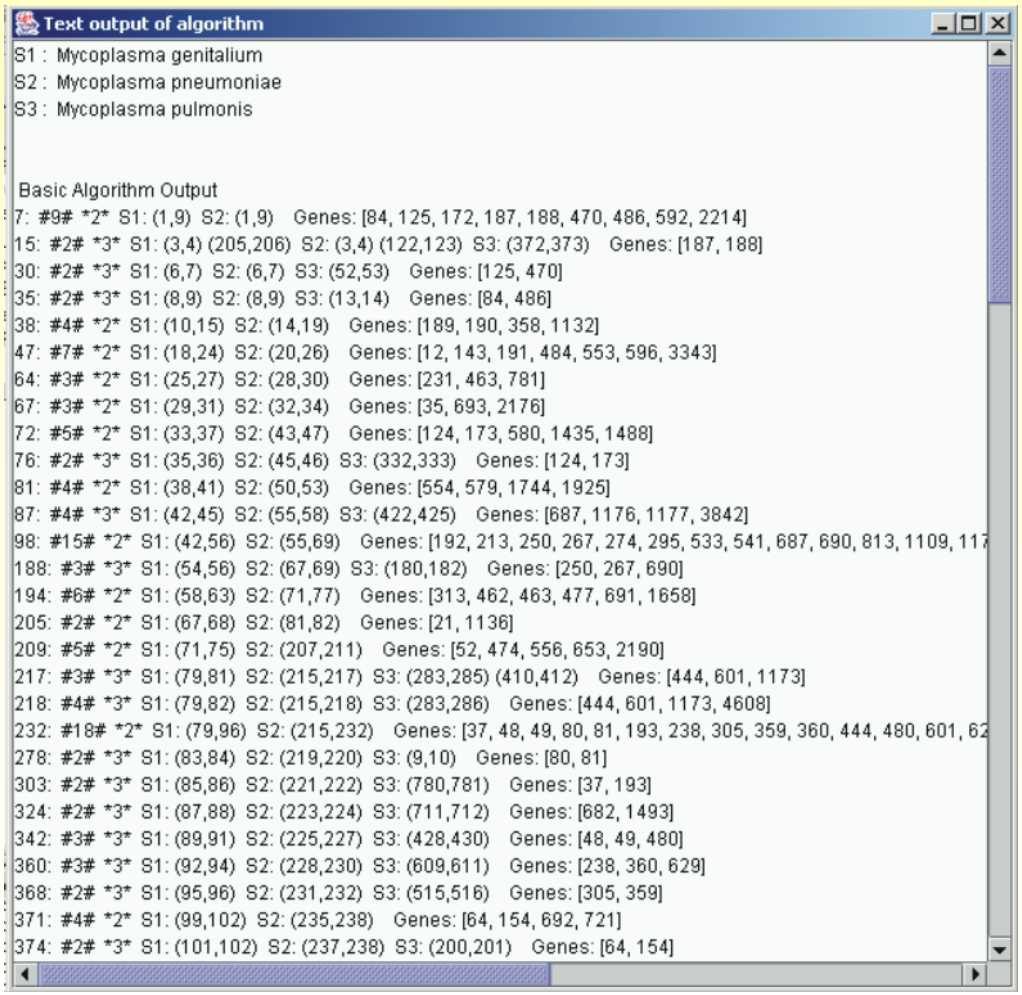
Location	Strand	Length	PID	Synonym	Code	COG	Product
1..2100	+	699	15605613	fusA	J	COG0480	elongation factor EF-G
17..3334	+	405	15605614	tufA1	J	COG0050	elongation factor EF-Tu
46..3660	+	104	15605615	rpsJ	J	COG0051	ribosomal protein S10
3665..4390	+	241	15605616	rplC	J	COG0087	ribosomal protein L03
4387..4986	+	199	15605617	rplD	J	COG0088	ribosomal protein L04
4990..5301	+	103	15605618	rplW	J	COG0089	ribosomal protein L23
5313..6227	+	304	15605619	rplB	J	COG0090	ribosomal protein L02
6340..6900	+	186	15605620	rpsS	J	COG0185	ribosomal protein S19
7018..7314	+	98	15605621	rplV	J	COG0091	ribosomal protein L22



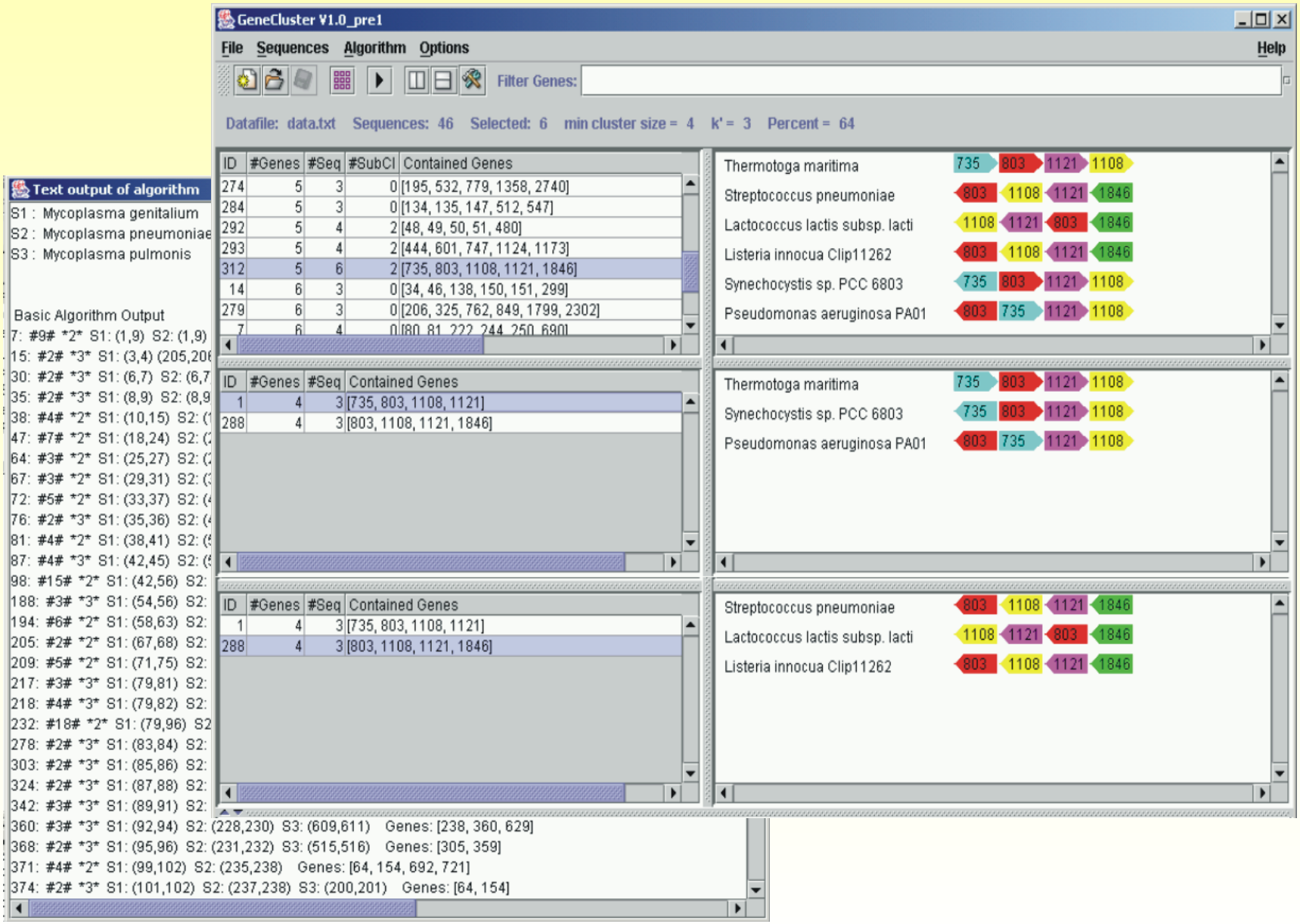
Experimental results. Application to 43 bacterial genomes



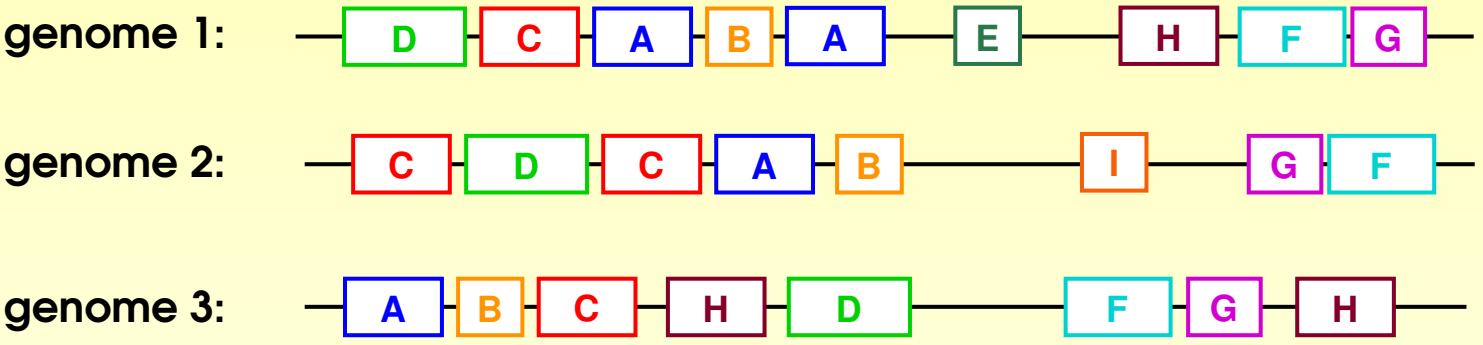
Experimental results. Graphical inspection of gene clusters: GECKO (bibiserv.techfak.uni-bielefeld.de/gecko)



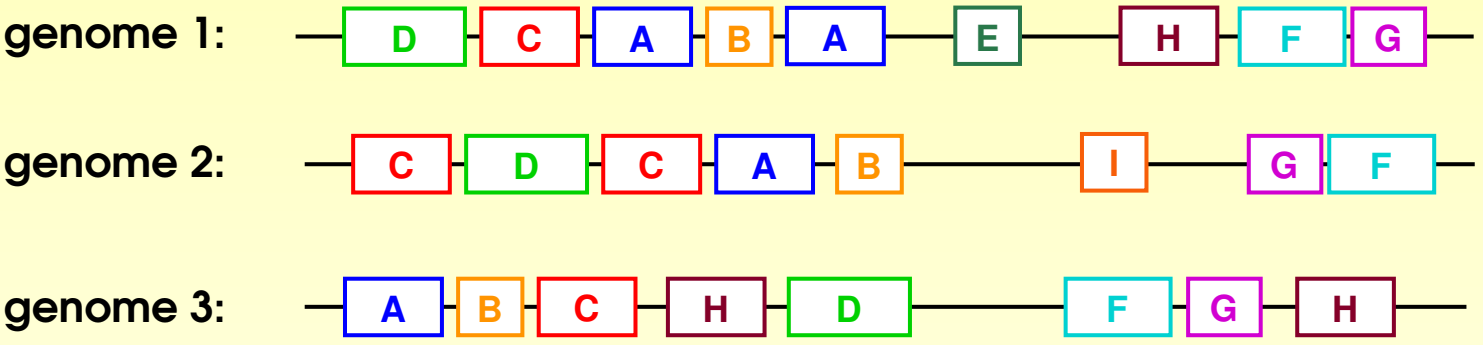
Experimental results. Graphical inspection of gene clusters: GECKO (bibiserv.techfak.uni-bielefeld.de/gecko)



Approximate Gene Clusters

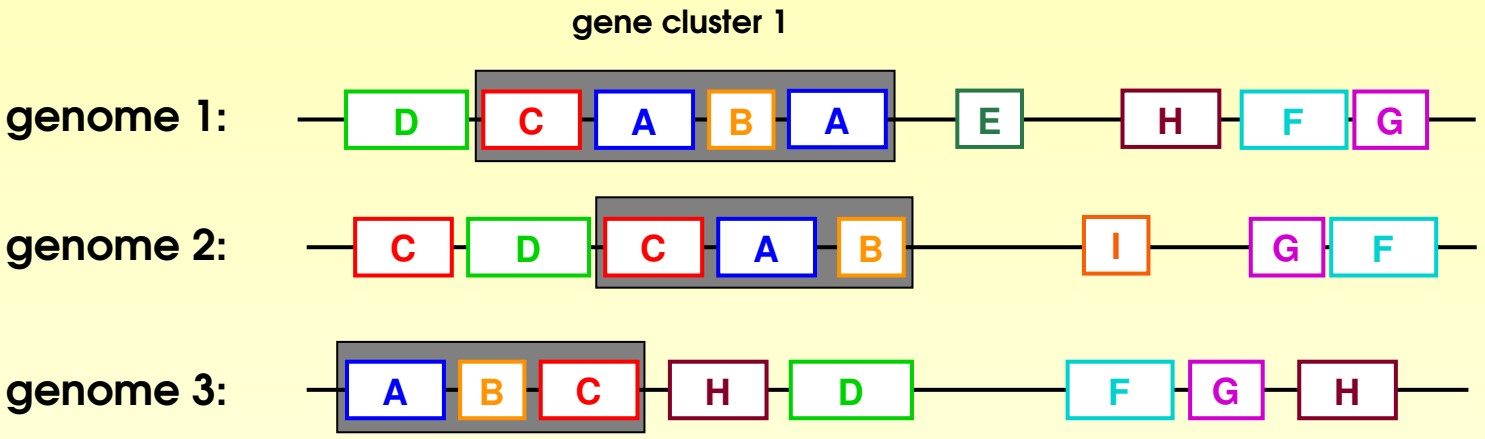


Approximate Gene Clusters



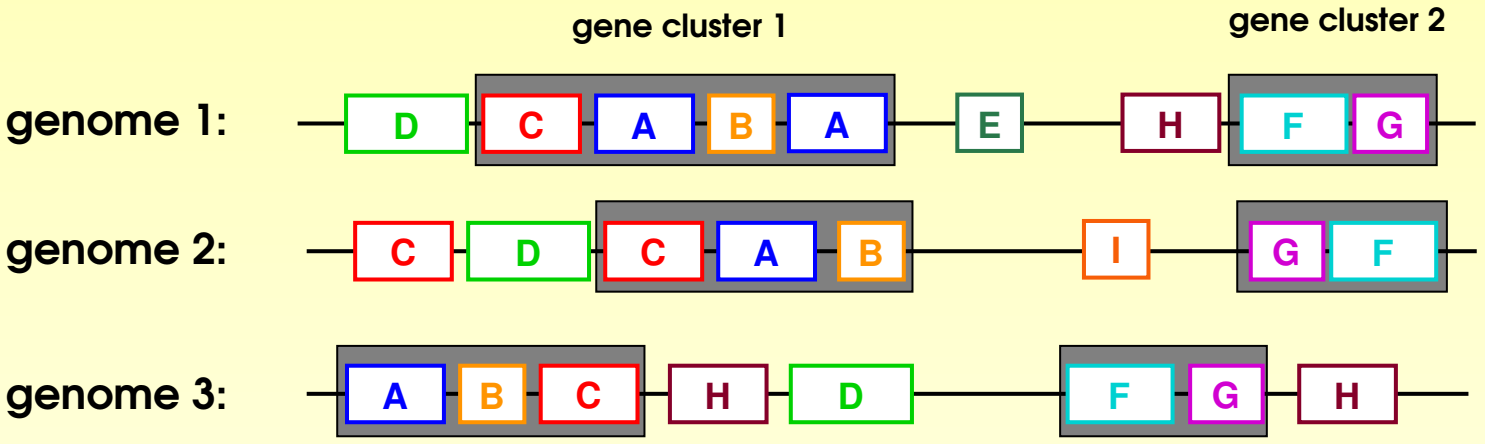
- (perfect) gene cluster:
  - set of genes occurring *en bloc* in multiple genomes
  - gene order within blocks not considered
  - multiple occurrences of genes possible

Approximate Gene Clusters



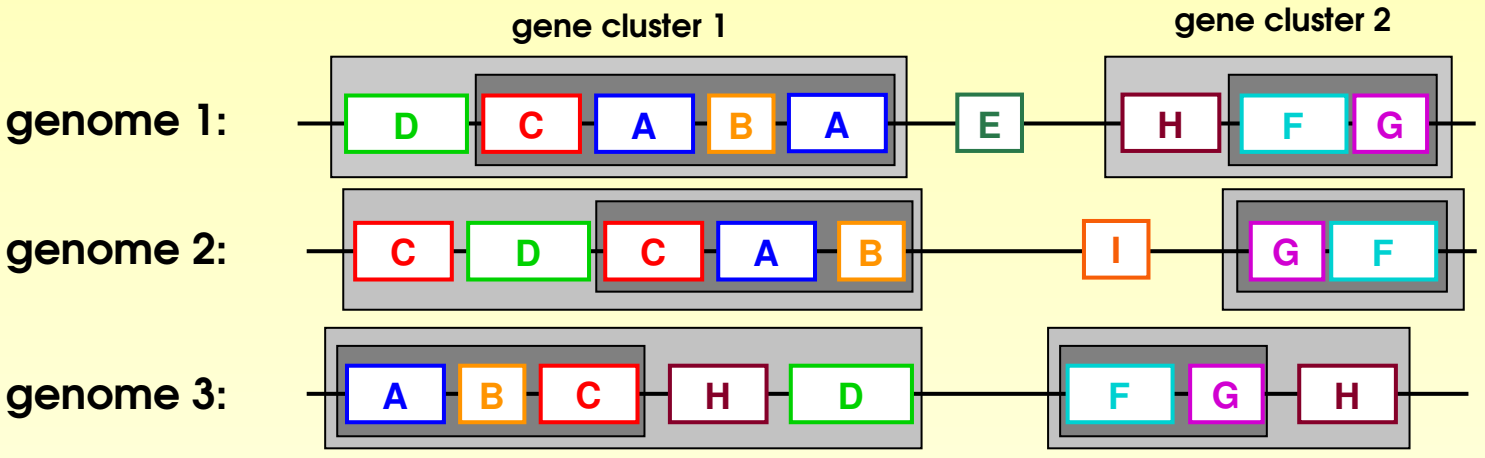
- (perfect) gene cluster:
  - set of genes occurring *en bloc* in multiple genomes
  - gene order within blocks not considered
  - multiple occurrences of genes possible

Approximate Gene Clusters



- (perfect) gene cluster:
  - set of genes occurring *en bloc* in multiple genomes
  - gene order within blocks not considered
  - multiple occurrences of genes possible

Approximate Gene Clusters



- (perfect) gene cluster:
  - set of genes occurring *en bloc* in multiple genomes
  - gene order within blocks not considered
  - multiple occurrences of genes possible
- approximate gene cluster:
  - additional and missing genes

Models for Approximate Gene Clusters

- *r*-window model (Friedman and Hughes, 2001)
  - fixed block size
  - pairwise comparison in polynomial time
- max-gap model (Bergeron, Corteel and Raffinot, 2002)
  - upper bound for insertion length
  - pairwise comparison in polynomial time
  - exponential in number of compared sequences
- approximate gene clusters (Rahmann and Klau, 2006)
  - very general cluster model, subsumes most other models
  - ILP approach
- **median gene clusters** (Böcker, Jahn, Mixtacki and JS, 2008)

Median of Character Sets

Sequence 1: . . . . . 1 2 3 4 5 6 7 . . . . . .
Sequence 2: . . . 1 8 3 4 5 2 1 6 7 9 . . . . . .
Sequence 3: . . . . . 8 3 7 5 1 . . .
Sequence 4: . . . 3 1 4 3 . . . . . .
Sequence 5: . . . . . 2 4 5 9 2 7 3 . . .

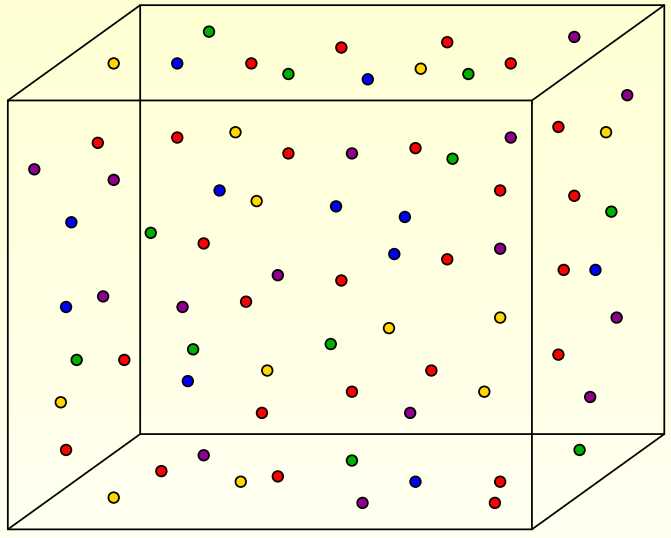
Σ :	1	2	3	4	5	6	7	8	9
$CS(S_1(i_1, j_1))$	1	1	1	1	1	1	1	0	0
$CS(S_2(i_2, j_2))$	1	1	1	1	1	1	1	1	1
$CS(S_3(i_3, j_3))$	1	0	1	0	1	0	1	1	0
$CS(S_4(i_4, j_4))$	1	0	1	1	0	0	0	0	0
$CS(S_5(i_5, j_5))$	0	1	1	1	1	0	1	0	1

Median of Character Sets

Sequence 1: . . . . . 1 2 3 4 5 6 7 . . . . . .
Sequence 2: . . . 1 8 3 4 5 2 1 6 7 9 . . . . . .
Sequence 3: . . . . . 8 3 7 5 1 . . .
Sequence 4: . . . 3 1 4 3 . . . . . .
Sequence 5: . . . . . 2 4 5 9 2 7 3 . . .

Σ :	1	2	3	4	5	6	7	8	9
$CS(S_1(i_1, j_1))$	1	1	1	1	1	1	1	0	0
$CS(S_2(i_2, j_2))$	1	1	1	1	1	1	1	1	1
$CS(S_3(i_3, j_3))$	1	0	1	0	1	0	1	1	0
$CS(S_4(i_4, j_4))$	1	0	1	1	0	0	0	0	0
$CS(S_5(i_5, j_5))$	0	1	1	1	1	0	1	0	1

Space of Character Sets: $\{0, 1\}^{|\Sigma|}$

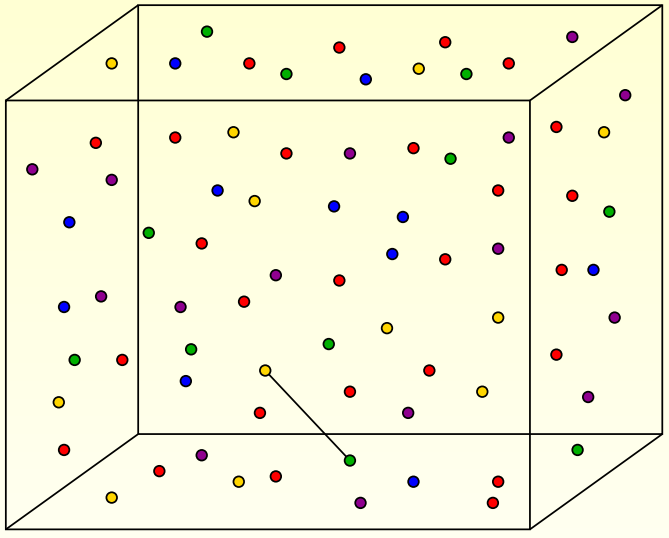


Median of Character Sets

Sequence 1: . . . . . 1 2 3 4 5 6 7 . . . . . .
Sequence 2: . . . 1 8 3 4 5 2 1 6 7 9 . . . . . .
Sequence 3: . . . . . 8 3 7 5 1 . . .
Sequence 4: . . . 3 1 4 3 . . . . . .
Sequence 5: . . . . . 2 4 5 9 2 7 3 . . .

Σ :	1	2	3	4	5	6	7	8	9
$CS(S_1(i_1, j_1))$	1	1	1	1	1	1	1	0	0
$CS(S_2(i_2, j_2))$	1	1	1	1	1	1	1	1	1
$CS(S_3(i_3, j_3))$	1	0	1	0	1	0	1	1	0
$CS(S_4(i_4, j_4))$	1	0	1	1	0	0	0	0	0
$CS(S_5(i_5, j_5))$	0	1	1	1	1	0	1	0	1

Space of Character Sets: $\{0,1\}^{|\Sigma|}$

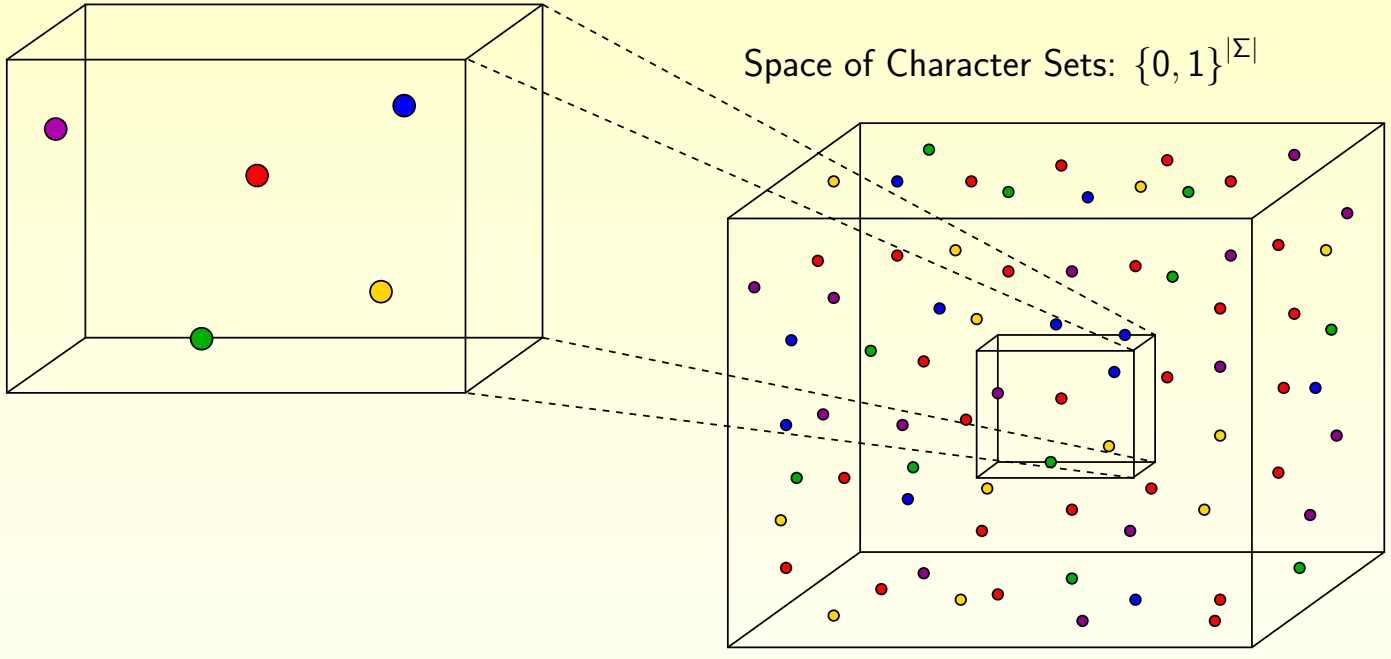


$$D_{sym}(C_1, C_2) = |C_1 \setminus C_2| + |C_2 \setminus C_1|$$

Median of Character Sets

Sequence 1: . . . . . 1 2 3 4 5 6 7 . . . . . .
Sequence 2: . . . 1 8 3 4 5 2 1 6 7 9 . . . . . .
Sequence 3: . . . . . 8 3 7 5 1 . . .
Sequence 4: . . . 3 1 4 3 . . . . . .
Sequence 5: . . . . . 2 4 5 9 2 7 3 . . .

Σ :	1	2	3	4	5	6	7	8	9
$CS(S_1(i_1, j_1))$	1	1	1	1	1	1	1	0	0
$CS(S_2(i_2, j_2))$	1	1	1	1	1	1	1	1	1
$CS(S_3(i_3, j_3))$	1	0	1	0	1	0	1	1	0
$CS(S_4(i_4, j_4))$	1	0	1	1	0	0	0	0	0
$CS(S_5(i_5, j_5))$	0	1	1	1	1	0	1	0	1

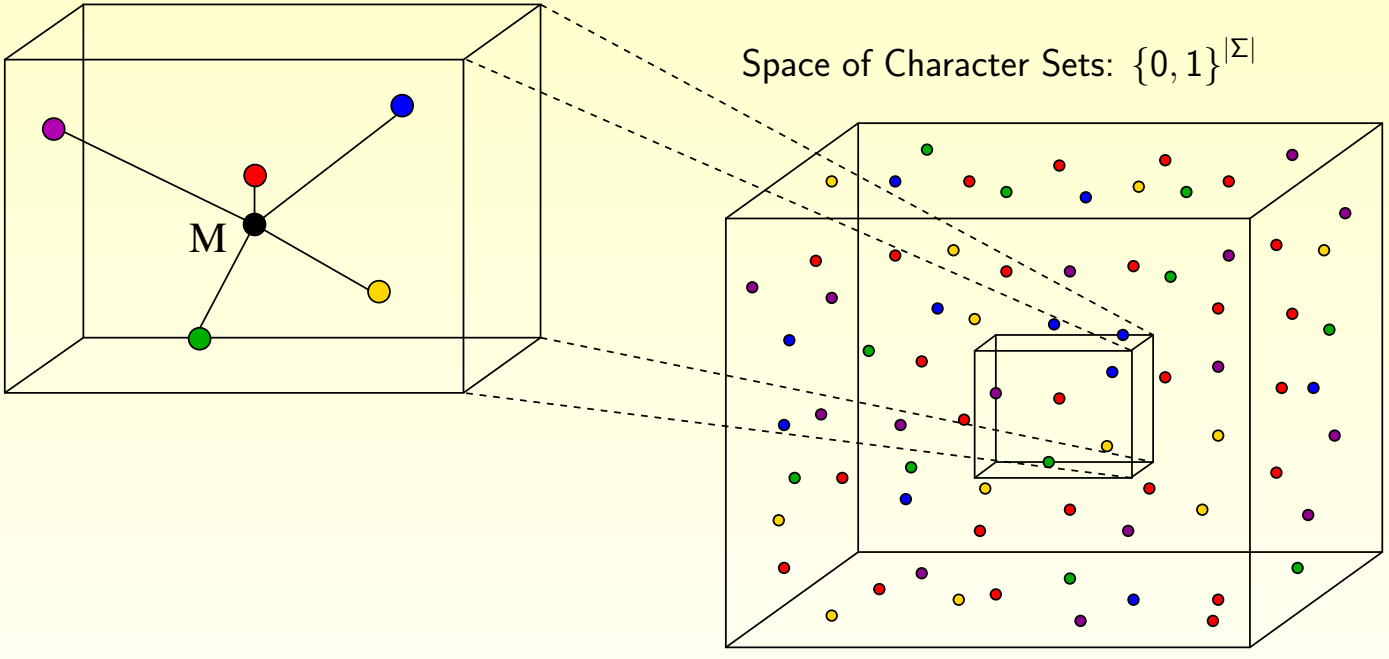


$$D_{sym}(C_1, C_2) = |C_1 \setminus C_2| + |C_2 \setminus C_1|$$

Median of Character Sets

Sequence 1: . . . . . 1 2 3 4 5 6 7 . . . . . .
Sequence 2: . . . 1 8 3 4 5 2 1 6 7 9 . . . . . .
Sequence 3: . . . . . 8 3 7 5 1 . . .
Sequence 4: . . . 3 1 4 3 . . . . . .
Sequence 5: . . . . . 2 4 5 9 2 7 3 . . .

Σ :	1	2	3	4	5	6	7	8	9
$CS(S_1(i_1, j_1))$	1	1	1	1	1	1	1	0	0
$CS(S_2(i_2, j_2))$	1	1	1	1	1	1	1	1	1
$CS(S_3(i_3, j_3))$	1	0	1	0	1	0	1	1	0
$CS(S_4(i_4, j_4))$	1	0	1	1	0	0	0	0	0
$CS(S_5(i_5, j_5))$	0	1	1	1	1	0	1	0	1

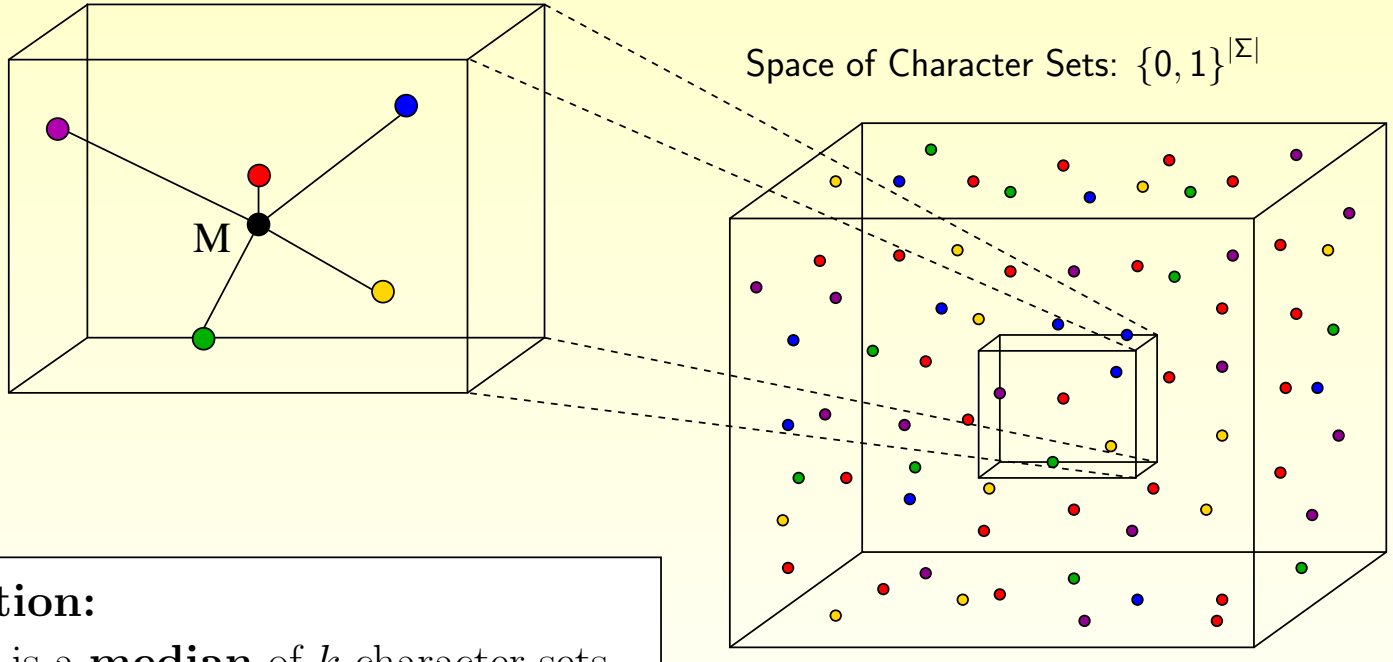


$$D_{sym}(C_1, C_2) = |C_1 \setminus C_2| + |C_2 \setminus C_1|$$

Median of Character Sets

Sequence 1: . . . . . 1 2 3 4 5 6 7 . . . . . .
Sequence 2: . . . 1 8 3 4 5 2 1 6 7 9 . . . . . .
Sequence 3: . . . . . 8 3 7 5 1 . . .
Sequence 4: . . . 3 1 4 3 . . . . . .
Sequence 5: . . . . . 2 4 5 9 2 7 3 . . .

Σ :	1	2	3	4	5	6	7	8	9
$CS(S_1(i_1, j_1))$	1	1	1	1	1	1	1	0	0
$CS(S_2(i_2, j_2))$	1	1	1	1	1	1	1	1	1
$CS(S_3(i_3, j_3))$	1	0	1	0	1	0	1	1	0
$CS(S_4(i_4, j_4))$	1	0	1	1	0	0	0	0	0
$CS(S_5(i_5, j_5))$	0	1	1	1	1	0	1	0	1



Definition:

$M \subseteq \Sigma$ is a **median** of k character sets $C_1, \ldots, C_k \subseteq \Sigma$ iff for all $C \subseteq \Sigma$:

$\Sigma_{\ell=1}^{k} D_{sym}(M, C_\ell) \leq \Sigma_{\ell=1}^{k} D_{sym}(C, C_\ell)$

$D_{sym}(C_1, C_2) = |C_1 \setminus C_2| + |C_2 \setminus C_1|$

Problem Statement

Given:

- sequences $S_1, \ldots, S_k$ over alphabet Σ
- s (minimum cluster size)
- δ (distance threshold)

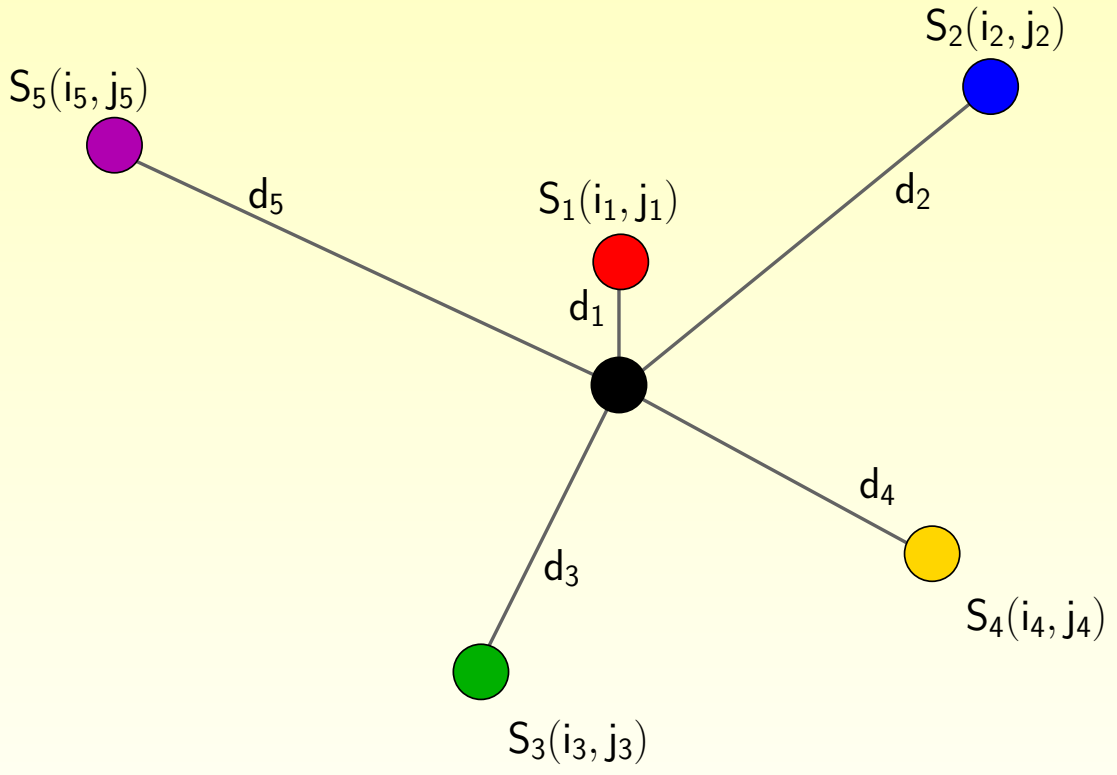
Wanted: all $M \subseteq \Sigma$ with

- M is a median for some $S_1[i_1, j_1], ..., S_k[i_k, j_k]$
- $\sum_{\ell=1}^{k} D\big(M, \mathcal{CS}(S[i_\ell, j_\ell])\big) \leq \delta$
- $|M| \geq s$

Such a set M is called a median gene cluster of $S_1, \ldots, S_k$.

Main Idea: Search Space Reduction

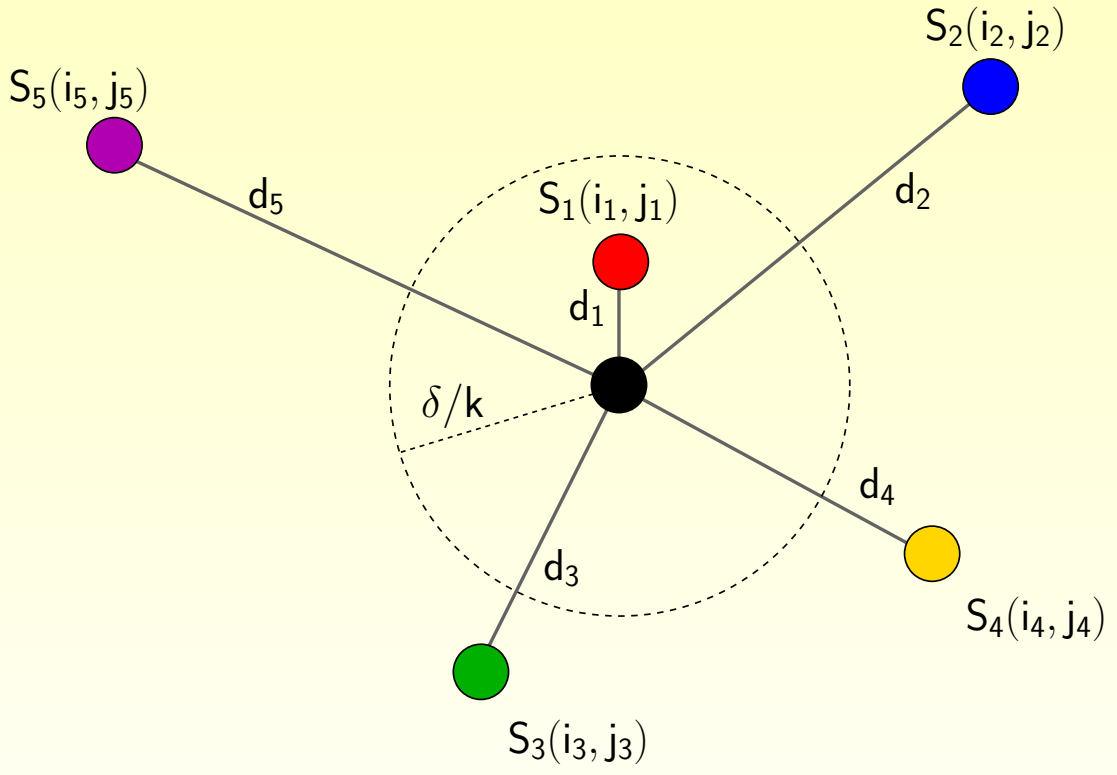
- search space: all $\mathcal{O}(n^{2k})$ combinations of substrings of $S_1, ..., S_k$
- cluster filter approach:



- median distance threshold: $\sum_{\ell=1}^{k} d_\ell \leq \delta$

Main Idea: Search Space Reduction

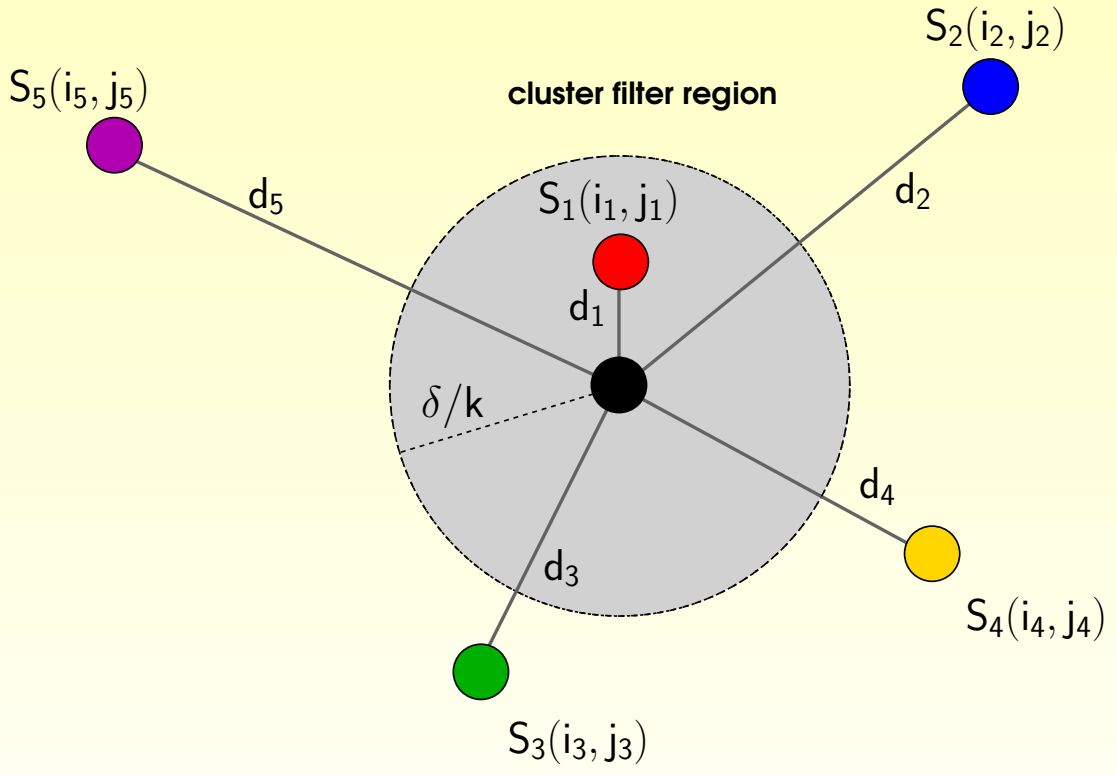
- search space: all $\mathcal{O}(n^{2k})$ combinations of substrings of $S_1, ..., S_k$
- cluster filter approach:



- median distance threshold: $\sum_{\ell=1}^{k} d_\ell \leq \delta$

Main Idea: Search Space Reduction

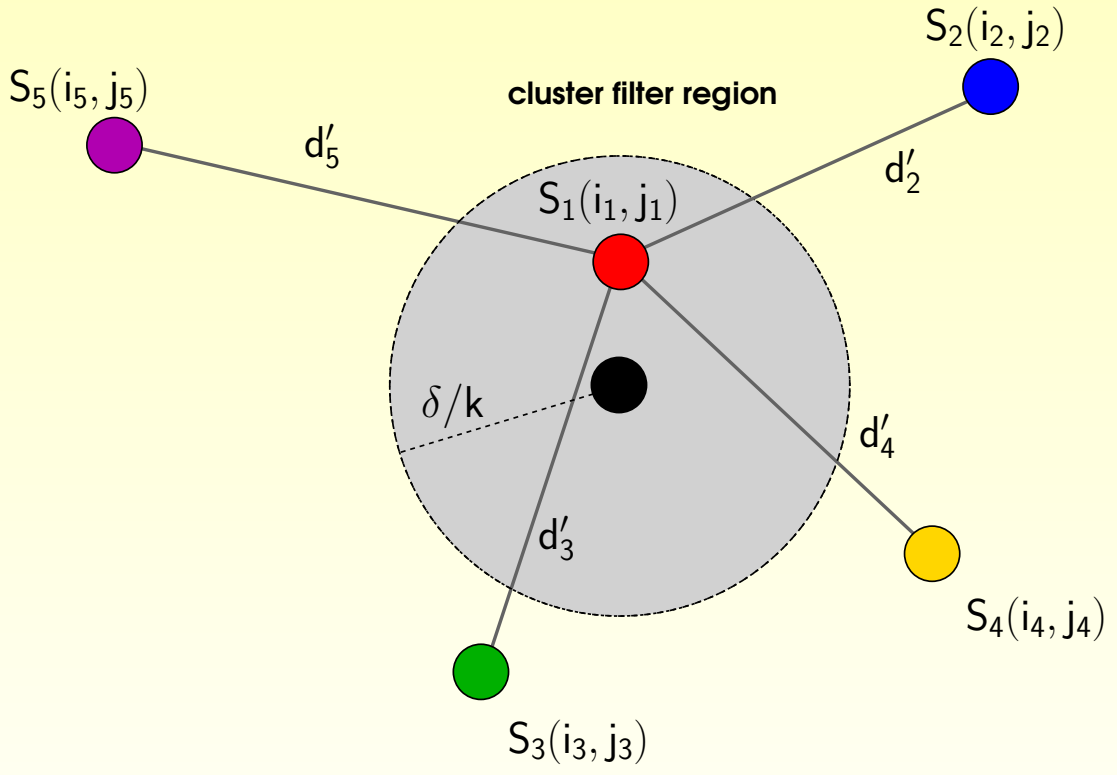
- search space: all $\mathcal{O}(n^{2k})$ combinations of substrings of $S_1, ..., S_k$
- cluster filter approach:



- median distance threshold: $\sum_{\ell=1}^{k} d_\ell \leq \delta$

Main Idea: Search Space Reduction

- search space: all $\mathcal{O}(n^{2k})$ combinations of substrings of $S_1, ..., S_k$
- cluster filter approach:



- median distance threshold: $\sum_{\ell=1}^{k} d_\ell \leq \delta$
- cluster filter distance threshold: $\sum_{\ell=1}^{k} d'_\ell \leq 2\frac{k-1}{k}\delta$

3-Step Approach to Median Gene Cluster Computation

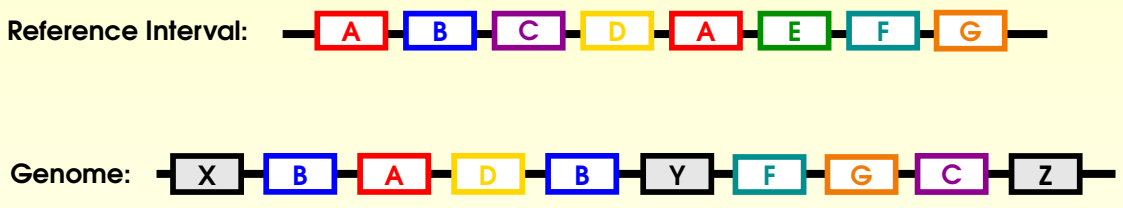
- Step 1: compute the set of all cluster filters C for $S_1, \ldots, S_k$
- Step 2: compute for each cluster filter C all combinations with substrings from the other sequences for which:

$$\sum_{\ell=1}^{k} D\big(C, \mathcal{CS}(S_\ell[i_\ell, j_\ell])\big) \;\leq\; 2\frac{k-1}{k}\delta$$

- Step 3:
  - computation of median(s) of each k-tuple (from Step 2)
  - comparison of median distance with distance threshold

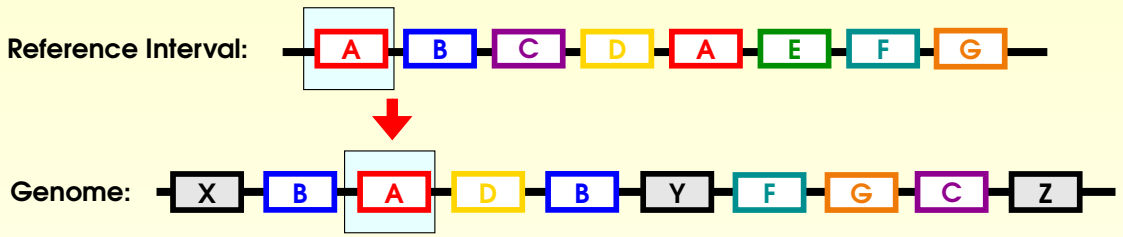
Step 1: Computation of Cluster Filters

- naive approach: testing all $\mathcal{O}(k^2 n^4)$ combinations of substrings
- our approach: extension of the *Connecting Intervals Algorithm* (Schmidt and JS, 2004)



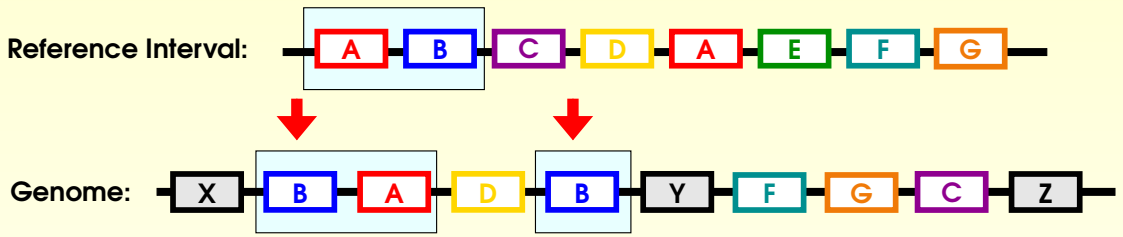
Step 1: Computation of Cluster Filters

- naive approach: testing all $\mathcal{O}(k^2 n^4)$ combinations of substrings
- our approach: extension of the *Connecting Intervals Algorithm* (Schmidt and JS, 2004)



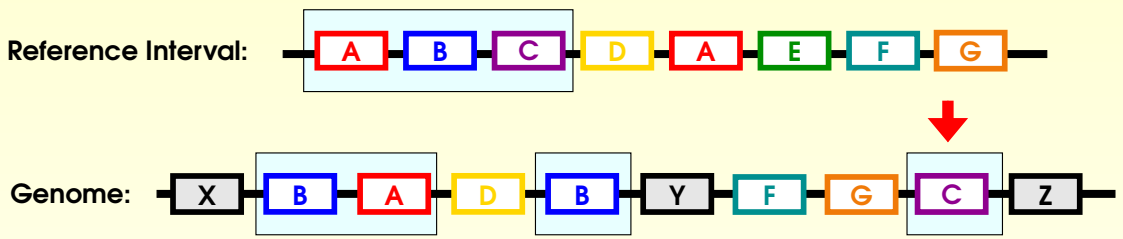
Step 1: Computation of Cluster Filters

- naive approach: testing all $\mathcal{O}(k^2 n^4)$ combinations of substrings
- our approach: extension of the *Connecting Intervals Algorithm* (Schmidt and JS, 2004)



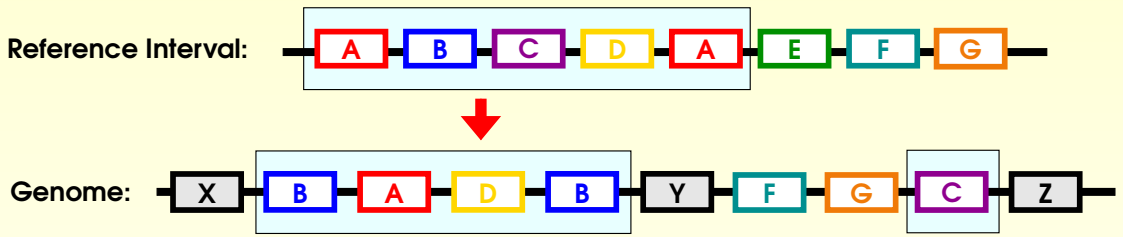
Step 1: Computation of Cluster Filters

- naive approach: testing all $\mathcal{O}(k^2 n^4)$ combinations of substrings
- our approach: extension of the *Connecting Intervals Algorithm* (Schmidt and JS, 2004)



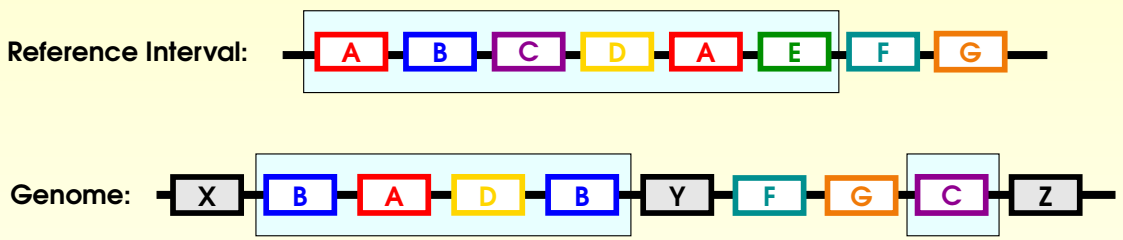
Step 1: Computation of Cluster Filters

- naive approach: testing all $\mathcal{O}(k^2 n^4)$ combinations of substrings
- our approach: extension of the *Connecting Intervals Algorithm* (Schmidt and JS, 2004)



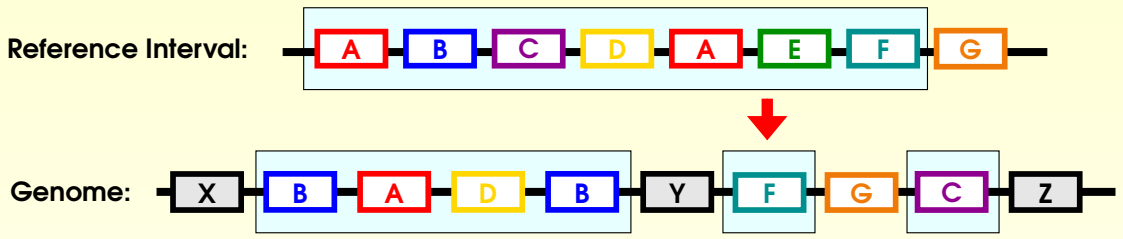
Step 1: Computation of Cluster Filters

- naive approach: testing all $\mathcal{O}(k^2 n^4)$ combinations of substrings
- our approach: extension of the *Connecting Intervals Algorithm* (Schmidt and JS, 2004)



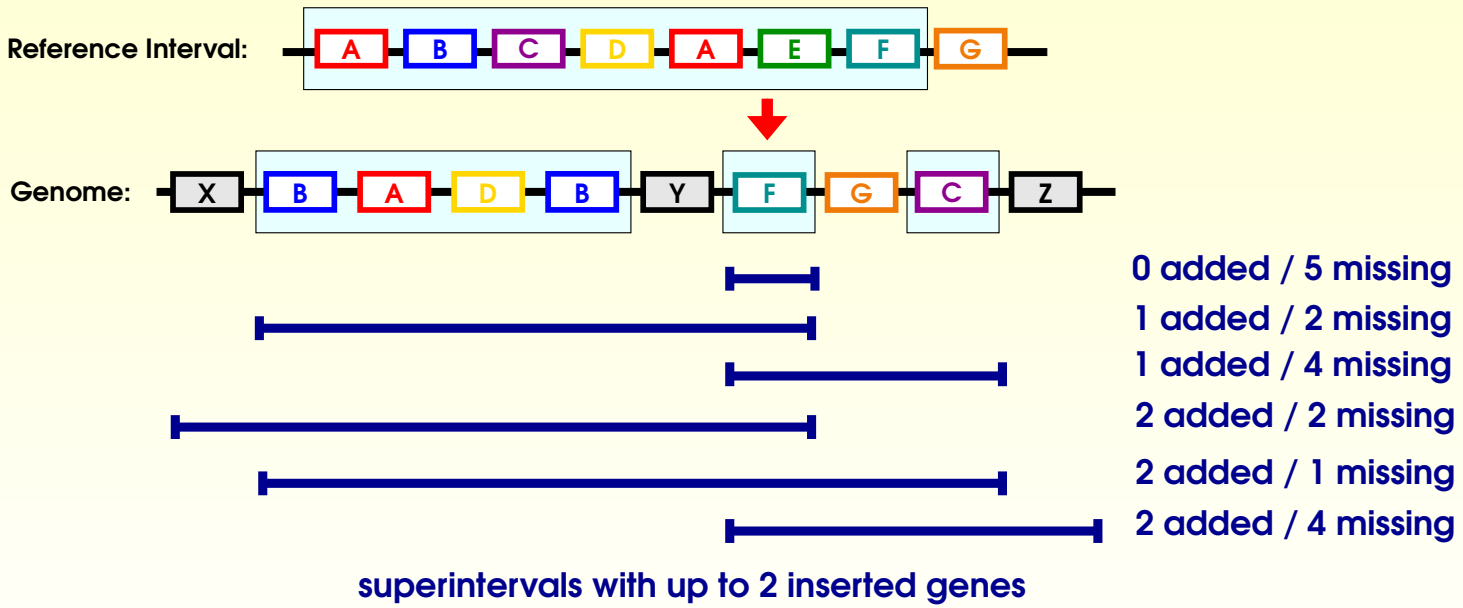
Step 1: Computation of Cluster Filters

- naive approach: testing all $\mathcal{O}(k^2 n^4)$ combinations of substrings
- our approach: extension of the *Connecting Intervals Algorithm* (Schmidt and JS, 2004)



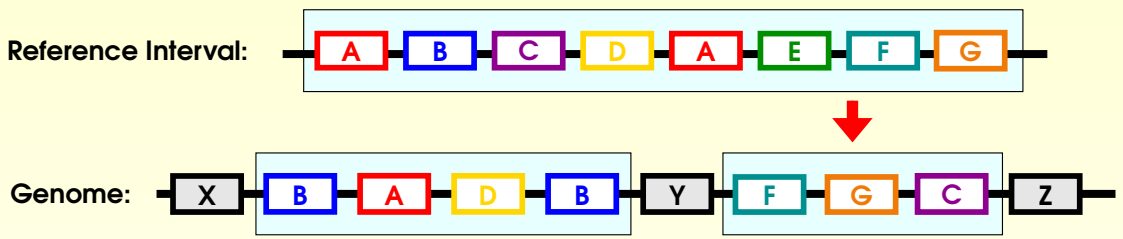
Step 1: Computation of Cluster Filters

- naive approach: testing all $\mathcal{O}(k^2 n^4)$ combinations of substrings
- our approach: extension of the *Connecting Intervals Algorithm* (Schmidt and JS, 2004)



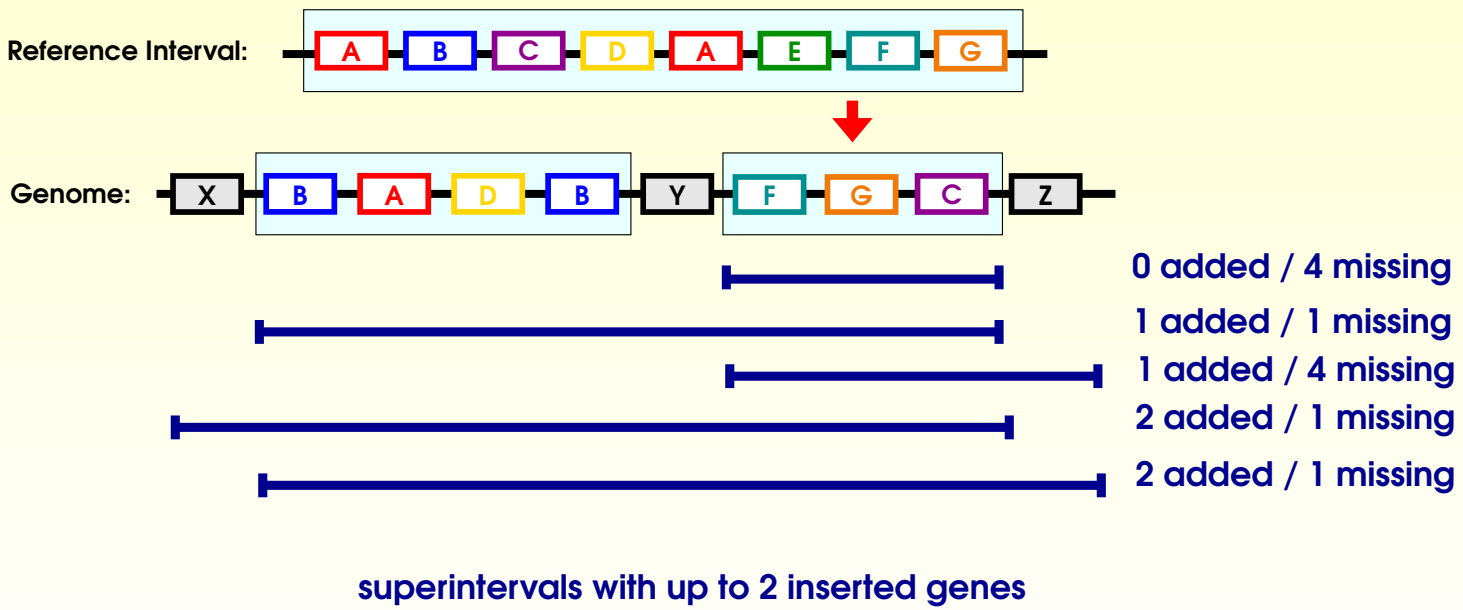
Step 1: Computation of Cluster Filters

- naive approach: testing all $\mathcal{O}(k^2 n^4)$ combinations of substrings
- our approach: extension of the *Connecting Intervals Algorithm* (Schmidt and JS, 2004)



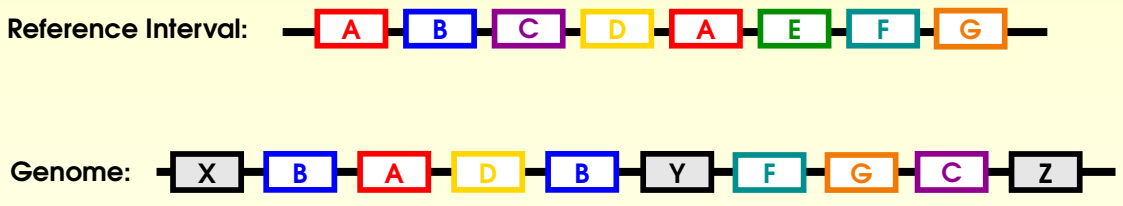
Step 1: Computation of Cluster Filters

- naive approach: testing all $\mathcal{O}(k^2 n^4)$ combinations of substrings
- our approach: extension of the *Connecting Intervals Algorithm* (Schmidt and JS, 2004)



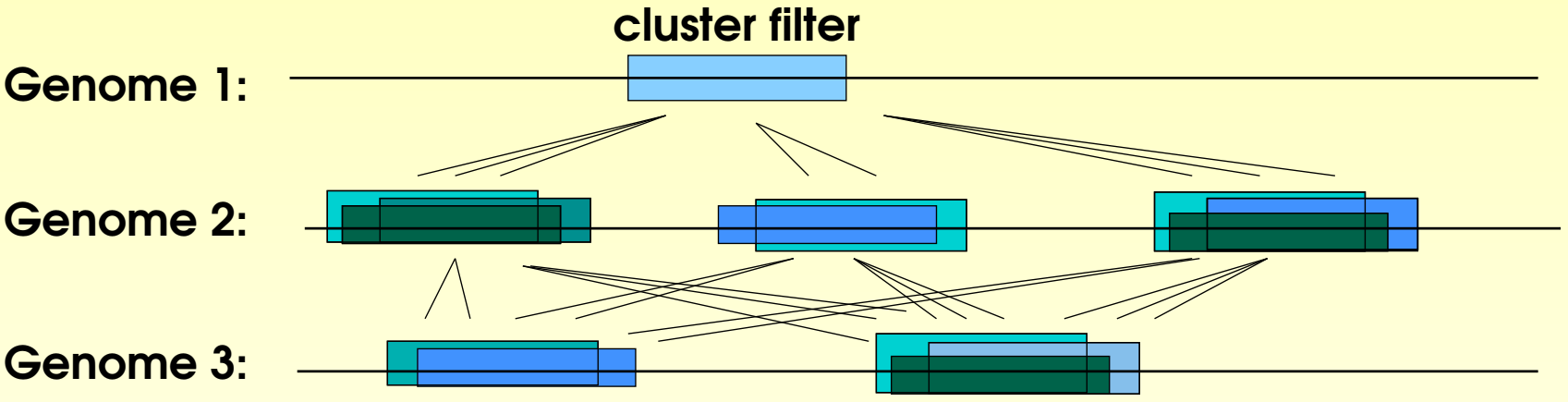
Step 1: Computation of Cluster Filters

- naive approach: testing all $\mathcal{O}(k^2 n^4)$ combinations of substrings
- our approach: extension of the *Connecting Intervals Algorithm* (Schmidt and JS, 2004)



- runtime: $\mathcal{O}(k^2 n^2(1+\delta^2))$, space: $\mathcal{O}(kn^2)$

Step 2: Computation of k-Tuples Containing a Cluster Filter



Idea: Build for each cluster filter C all combinations with substrings of the other sequences for which C is cluster filter

- possible combinatorial explosion $\mathcal{O}(n^k)$ (unlikely in practice)
- up to $\mathcal{O}(\delta^{2k})$ variants of a k-tuple

Step 3: Computation of Median of Each k-Tuple

- majority vote: $\mathcal{O}(k|\Sigma|)$ time and $\mathcal{O}(k|\Sigma|)$ space
- compare total distance with threshold δ

	A	B	C	D	E	F	G	H	I	J
Genome 1	1	1	1	1	0	1	0	1	1	0
Genome 2	1	1	0	1	0	1	1	1	1	1
Genome 3	1	1	1	1	1	1	0	1	0	1
Genome 4	1	1	1	1	1	1	1	1	1	0
Genome 5	1	0	1	1	0	1	0	1	1	1
	5	4	4	5	2	5	2	5	4	3

Median = { A B C D F H I J }

Step 3: Computation of Median of Each k-Tuple

- majority vote: $\mathcal{O}(k|\Sigma|)$ time and $\mathcal{O}(k|\Sigma|)$ space
- compare total distance with threshold δ

	A	B	C	D	E	F	G	H	I	J
Genome 1	1	1	1	1	0	1	0	1	1	0
Genome 2	1	1	0	1	0	1	1	1	1	1
Genome 3	1	1	1	1	1	1	0	1	0	1
Genome 4	1	1	1	1	1	1	1	1	1	0
Genome 5	1	0	1	1	0	1	0	1	1	1
	5	4	4	5	2	5	2	5	4	3

Median = { A B C D F H I J }

Algorithm Summary

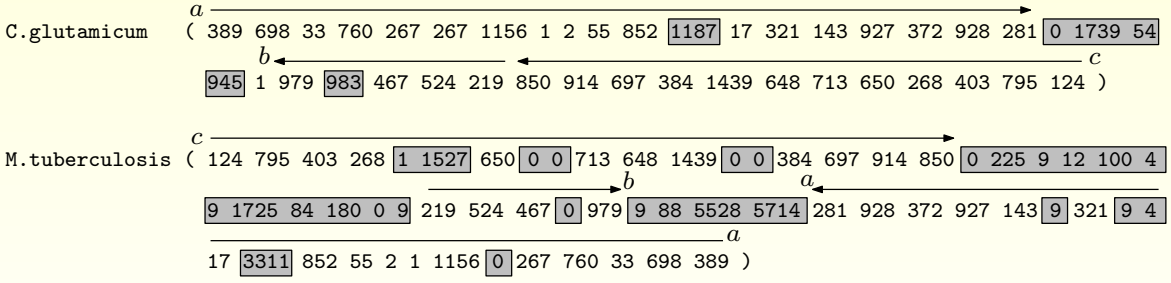
- Step 1: cluster filter computation $\mathcal{O}(k^2 n^2 (1 + \delta^2))$
- Step 2: combination of each cluster filter C with substrings from other sequences with

$$\sum_{\ell=1}^{k} D(C, \mathcal{CS}(S_\ell[i_\ell, j_\ell])) \leq 2\frac{k-1}{k}\delta$$

  possible combinatorial explosion: $\mathcal{O}(n^k)$
- Step 3:
  - computation of median of each such k-tuple $\mathcal{O}(k|\Sigma|)$
  - comparison: median distance vs. distance threshold δ

Experimental Results

- comparison with ILP approach (Rahmann and Klau 2006)
- dataset: genomes of *C. glutamicum* and *M. tuberculosis*
- detection of gene cluster containing 51 genes
- runtime of ILP program: $>$ 1h
- runtime of our approach: 17 sec. (on slower processor)



Experimental Results

- application to gene cluster detection in multiple genomes
- five γ-proteobacteria:
  - *Buchnera aphidicola APS (NC_002528)*
  - *Escherichia coli K12 (NC_000913)*
  - *Haemophilus influenzae Rd (NC_000907)*
  - *Pasteurella multocida Pm70 (NC_002663)*
  - *Xylella fastidiosa 9a5c (NC_002488)*

	δ=0	δ=1	δ=5	δ=8	δ=12
s=10	6	124	252	329	406
s=15	0	42	127	165	181
s=25	0	0	10	10	10

\# median gene clusters

	δ=0	δ=1	δ=5	δ=8	δ=12
s=10	6.9	7.4	20.0	103.4	1610.0
s=15	6.9	7.3	15.1	59.7	1148.5
s=25	6.9	7.1	10.5	17.6	630.0

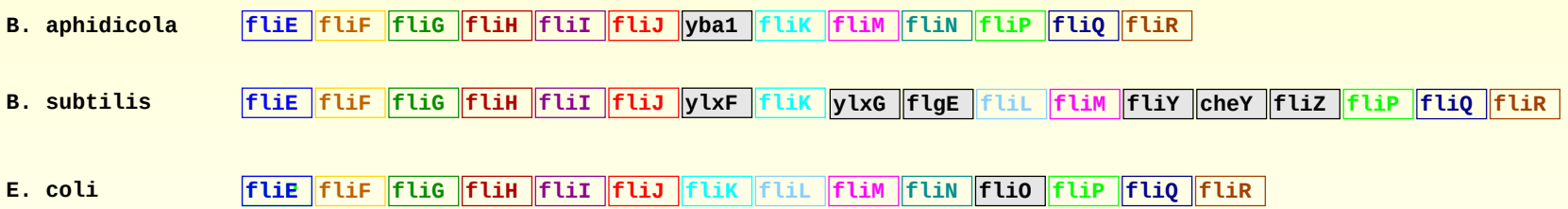
computation time (seconds)

Experimental Results

Sample gene clusters detected by our method:

- ATP synthesis

- flagellar biosynthesis



- cell division and cell wall biosynthesis

Algorithmic results

- Find all *common intervals* of k permutations in $\mathcal{O}(kn + |\text{output}|)$ time.
- Find all *common intervals* of k sequences in $\mathcal{O}(kn^2)$ time.
- Find all *median gene clusters* of k sequences in $\mathcal{O}(k^2n^2(1+\delta^2) + n^k + k|\Sigma|)$ time

Conclusion

Points raised:

- Comparative genomics can help in functional genome annotation
- Conserved regions in genomes have a static and a dynamic aspect
- Interesting combinatorics in Bioinformatics

Next steps:

- Statistical assessment of gene clusters
- Patterns in overlapping gene clusters
- Application to more data

Acknowledgments

- Sebastian Böcker (Bielefeld/Jena)
- Steffen Heber (Raleigh)
- **Katharina Jahn** (Bielefeld)
- Hannes Luz (Berlin)
- Julia Mixtacki (Bielefeld)
- Mathieu Raffinot (Paris/Moscow)
- Sven Rahmann (Bielefeld/Dortmund)
- **Thomas Schmidt** (Bielefeld/München)

References:

- Heber & JS, Finding all Common Intervals of k Permutations, Proc. CPM 2001.
- Heber & JS, Algorithms for Finding Gene Clusters, Proc. WABI 2001.
- Schmidt & JS, Quadratic Time Algorithms for Finding Common Intervals in Two and More Sequences, Proc. CPM 2004.
- Böcker, Jahn, Mixtacki & JS, Computation of Median Gene Clusters, Proc. RECOMB 2008.

The end.