# RNA-Sequencing

## Lecture 2: Read-mapping through Filter-based approaches

Nicolas Balcazar Corinna Blasse An Duc Dang
Hannes Hauswedell Sebastian Thieme

Advanced Algorithms for Bioinformatics (P4)
K. Reinert and S. Andreotti

SoSe 2010
FU Berlin

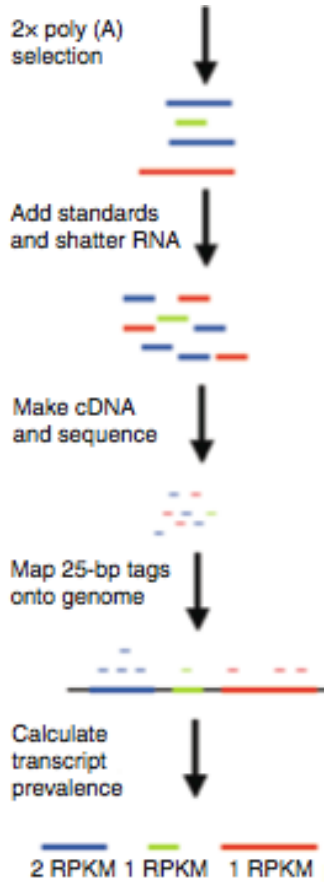
May 26, 2010

# Outline

# Outline

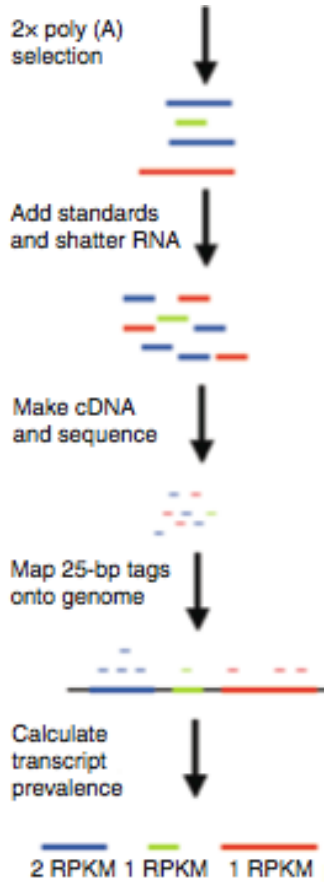
# Pipeline

- Poly(A)-selection of RNA
- Fragmentation of RNA to an average length
- Conversion into cDNA
- Sequencing
- Mapping of the reads onto the genome



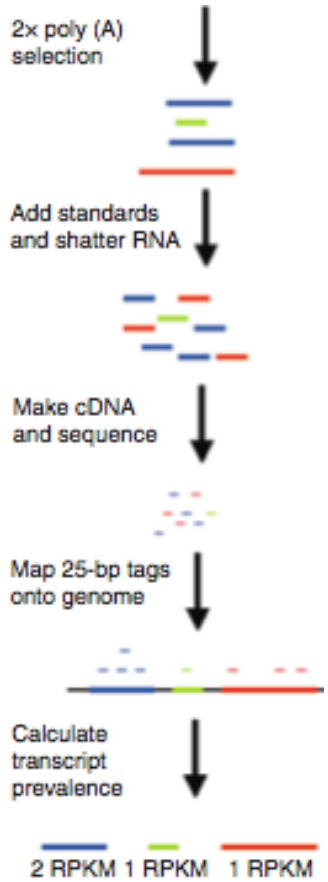
# Pipeline

- Poly(A)-selection of RNA
- Fragmentation of RNA to an average length
- Conversion into cDNA
- Sequencing
- Mapping of the reads onto the genome



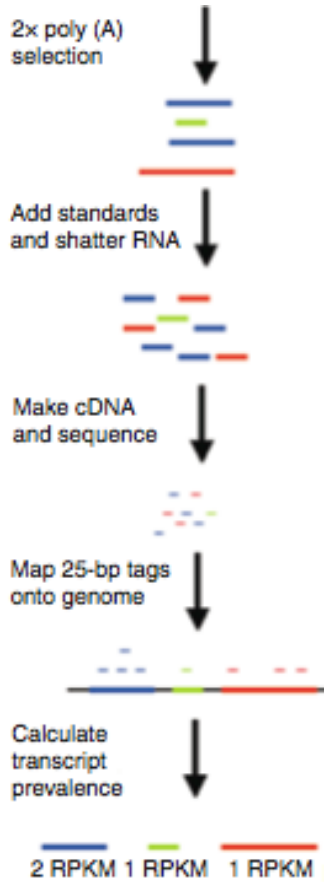
# Pipeline

- Poly(A)-selection of RNA
- Fragmentation of RNA to an average length
- Conversion into cDNA
- Sequencing
- Mapping of the reads onto the genome



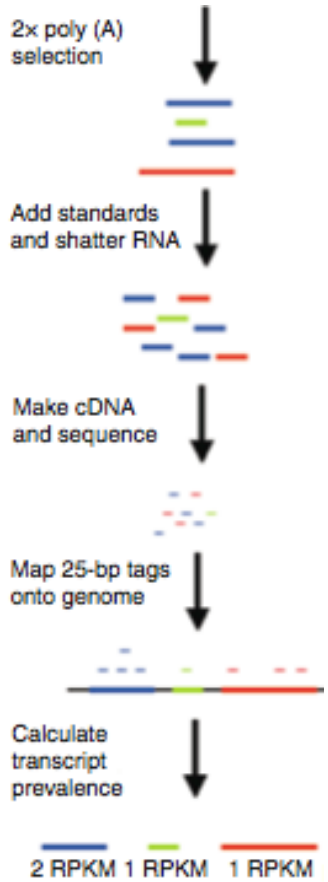
# Pipeline

- Poly(A)-selection of RNA
- Fragmentation of RNA to an average length
- Conversion into cDNA
- Sequencing
- Mapping of the reads onto the genome



# Pipeline

- Poly(A)-selection of RNA
- Fragmentation of RNA to an average length
- Conversion into cDNA
- Sequencing
- Mapping of the reads onto the genome

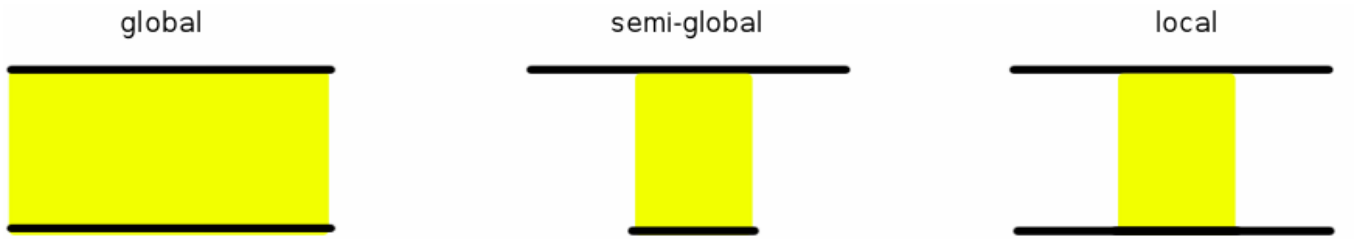


# Pipeline

- Poly(A)-selection of RNA
- Fragmentation of RNA to an average length
- Conversion into cDNA
- Sequencing
- Mapping of the reads onto the genome

# Mapping

- *Read-mapping* is the semi-global alignment of many (very) short sequences (reads) to a long sequence (the genome)
- Alignment: approximate string matching



# filtering vs non-filtering

**bowtie:** non-filtering

- BWT
- L to F mapping
- backtracking

**problem:**

- does not find all approximate matches
- increase in error rate $\Longrightarrow$ exponential increase in running time

# filtering vs non-filtering

bowtie: non-filtering

- BWT
- L to F mapping
- backtracking

problem:

- does not find all approximate matches
- increase in error rate $\Longrightarrow$ exponential increase in running time

# filtering vs non-filtering

**bowtie:** non-filtering

- BWT
- L to F mapping
- backtracking

**problem:**

- does not find all approximate matches
- increase in error rate $\Longrightarrow$ exponential increase in running time

# filtering vs non-filtering

filtering idea: faster to identify non-matching sections than to find a position of a match

- discard all parts, which do not contain a possible matching position
- retain all sub-strings which might contain a match

# filtering vs non-filtering

filtering idea: faster to identify non-matching sections than to find a position of a match

- discard all parts, which do not contain a possible matching position
- retain all sub-strings which might contain a match

# filtering vs non-filtering

Filtering sensitive to error level:

$$\alpha := \frac{k}{m} \qquad \text{(k = error; m = pattern length)}$$

Higher error level

- cost of verification increases
- reduction of filter efficiency

discard large segments of the text $\Rightarrow$ fast search
$\Rightarrow$ improve average-case performance

# filtering vs non-filtering

Filtering sensitive to error level:

$$\alpha := \frac{k}{m} \qquad \text{(k = error; m = pattern length)}$$

Higher error level

- cost of verification increases
- reduction of filter efficiency

discard large segments of the text $\Rightarrow$ fast search
$\Rightarrow$ improve average-case performance

# filtering vs non-filtering

Filtering sensitive to error level:

$$\alpha := \frac{k}{m} \qquad \text{(k = error; m = pattern length)}$$

Higher error level

- cost of verification increases
- reduction of filter efficiency

discard large segments of the text $\Rightarrow$ fast search
$\Rightarrow$ improve average-case performance

# Outline

# Basic principle

- $m$ objects and $n$ sets ($m < n$)
- each object is contained in one set

# Basic principle

- $m$ objects (pigeons) and $n$ sets (holes) ($m < n$)
- each object is contained in one set

# Outline

# Pigeonhole principle

- find all occurrences of a Pattern ($|P| = m$) in a text ($|T| = n$) with at most $k$ errors

holes: $k + 1$ pattern pieces

pigeons: $k$ errors

rule: one of the pattern pieces has to match without error

# Pigeonhole principle

- find all occurrences of a Pattern ($|P| = m$) in a text ($|T| = n$) with at most $k$ errors

holes: $k + 1$ pattern pieces

pigeons: $k$ errors

rule: one of the pattern pieces has to match without error

# Pigeonhole principle

## Lemma

*Let Occ match $P$ with $k$ errors, $P = p^1, \ldots, p^j$ be a concatenation of subpatterns, and $a_1, \ldots, a_j$ be* ***nonnegative*** *integers such that $A = \sum_{i=1}^{j} a_i$. Then, for some $i \in \{1, \ldots, j\}$, Occ includes a substring that matches $p^i$ with $\lfloor \frac{a_i k}{A} \rfloor$ errors.*

# Basic procedure

Divide: $k + 1$ pattern pieces

Search: simultaneously search with multi-pattern string matching algorithm

Verify: check the neighbourhood of their occurrence

# Basic procedure

Divide: $k + 1$ pattern pieces

Search: simultaneously search with multi-pattern string matching algorithm

Verify: check the neighbourhood of their occurrence

# Basic procedure

**Divide:** $k + 1$ pattern pieces

**Search:** simultaneously search with multi-pattern string matching algorithm

**Verify:** check the neighbourhood of their occurrence

# Verification

- $p^j[start : end]$ matches at $t[j : j + (end_i - start_i)]$
- occurrences are of length at most m+k
- occurrence can start at most $start_i - 1 + k$ before position $j$ in $t$
- can finish at most $m - end_i + k$ after $t[j + (end_i - start_i)]$

$\Rightarrow$ check the text area of
$t[j - (start_i - 1) - k : j + (m - start_i) + k] \Rightarrow$ length of text area $m + 2k$

# Verification

- $p^j[start : end]$ matches at $t[j : j + (end_i - start_i)]$
- occurrences are of length at most m+k
- occurrence can start at most $start_i - 1 + k$ before position $j$ in $t$
- can finish at most $m - end_i + k$ after $t[j + (end_i - start_i)]$

$\Rightarrow$ check the text area of
$t[j - (start_i - 1) - k : j + (m - start_i) + k] \Rightarrow$ length of text area $m + 2k$

# Verification

- $p^j[start : end]$ matches at $t[j : j + (end_i - start_i)]$
- occurrences are of length at most m+k
- occurrence can start at most $start_i - 1 + k$ before position $j$ in $t$
- can finish at most $m - end_i + k$ after $t[j + (end_i - start_i)]$

$\Rightarrow$ check the text area of
$t[j - (start_i - 1) - k : j + (m - start_i) + k] \Rightarrow$ length of text area $m + 2k$

# Verification

- $p^j[start : end]$ matches at $t[j : j + (end_i - start_i)]$
- occurrences are of length at most m+k
- occurrence can start at most $start_i - 1 + k$ before position $j$ in $t$
- can finish at most $m - end_i + k$ after $t[j + (end_i - start_i)]$

$\Rightarrow$ check the text area of
$t[j - (start_i - 1) - k : j + (m - start_i) + k] \Rightarrow$ length of text area $m + 2k$

# Verification

- $p^j[start : end]$ matches at $t[j : j + (end_i - start_i)]$
- occurrences are of length at most m+k
- occurrence can start at most $start_i - 1 + k$ before position $j$ in $t$
- can finish at most $m - end_i + k$ after $t[j + (end_i - start_i)]$

$\Rightarrow$ check the text area of
$t[j - (start_i - 1) - k : j + (m - start_i) + k]$ $\Rightarrow$ length of text area $m + 2k$

# Verification

- $p^j[start : end]$ matches at $t[j : j + (end_i - start_i)]$
- occurrences are of length at most m+k
- occurrence can start at most $start_i - 1 + k$ before position $j$ in $t$
- can finish at most $m - end_i + k$ after $t[j + (end_i - start_i)]$

$\Rightarrow$ check the text area of
$t[j - (start_i - 1) - k : j + (m - start_i) + k] \Rightarrow$ length of text area $m + 2k$

# An example

pattern: annual

text ($t$): any_annealing

error (k): 2

# An example

$t =$ any_annealing

Dividing: annual $\Rightarrow p^1 = an, p^2 = nu, p^3 = al$

Searching: $an$ in $t \Rightarrow$ pos $1, 5$
nu in $t \Rightarrow$ pos *None*
al in $t \Rightarrow$ pos $9$

Verification: three occurrences in $t \Rightarrow 9, 10, 11$

| $t[9]$ | $t[10]$ | $t[11]$ |
|---|---|---|
| annea- | anneal | anneali |
| annual | annual | annual- |

# An example

$t =$ any_annealing

Dividing: annual $\Rightarrow p^1 = an, p^2 = nu, p^3 = al$

Searching:
$an$ in $t \Rightarrow$ pos $1, 5$
nu in $t \Rightarrow$ pos *None*
al in $t \Rightarrow$ pos 9

Verification: three occurrences in $t \Rightarrow 9, 10, 11$

| $t[9]$ | $t[10]$ | $t[11]$ |
|---|---|---|
| annea- | anneal | anneali |
| annual | annual | annual- |

# An example

$t =$ any_annealing

Dividing: annual $\Rightarrow p^1 = an, p^2 = nu, p^3 = al$

Searching:
$an$ in $t \Rightarrow$ pos $1, 5$
nu in $t \Rightarrow$ pos *None*
al in $t \Rightarrow$ pos 9

Verification: three occurrences in $t \Rightarrow 9, 10, 11$

| $t[9]$ | $t[10]$ | $t[11]$ |
|---|---|---|
| annea- | anneal | anneali |
| annual | annual | annual- |

# An example

$t =$ any_annealing

Dividing: annual $\Rightarrow p^1 = an, p^2 = nu, p^3 = al$

Searching:
$an$ in $t \Rightarrow$ pos $1, 5$
nu in $t \Rightarrow$ pos *None*
al in $t \Rightarrow$ pos 9

Verification: three occurrences in $t \Rightarrow 9, 10, 11$

| $t[9]$ | $t[10]$ | $t[11]$ |
|---|---|---|
| annea- | anneal | anneali |
| annual | annual | annual- |

# An example

$t$ = any_annealing

**Dividing:** annual $\Rightarrow p^1 = an, p^2 = nu, p^3 = al$

**Searching:**
*an* in $t \Rightarrow$ pos $1, 5$
nu in $t \Rightarrow$ pos *None*
al in $t \Rightarrow$ pos 9

**Verification:** three occurrences in $t \Rightarrow 9, 10, 11$

| $t[9]$ | $t[10]$ | $t[11]$ |
|---|---|---|
| annea- | anneal | anneali |
| annual | annual | annual- |

# Problem

text2 ($t_2$): an_unusual_example_with_numerous_verification

- many verifications are unsuccessful
- repeated verification of the same sub-pattern

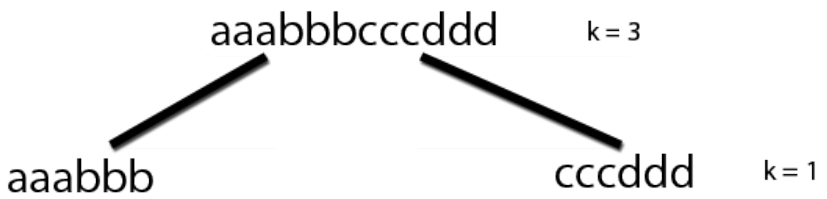
# Hierarchical approach

- divide the pattern into $k+1$ pieces hierarchically
  1. split the pattern in two pieces and search for each piece with $k = \lfloor \frac{k}{2} \rfloor$
  2. halves are recursively split and searched until error rate reaches zero

aaabbbcccddd k = 3

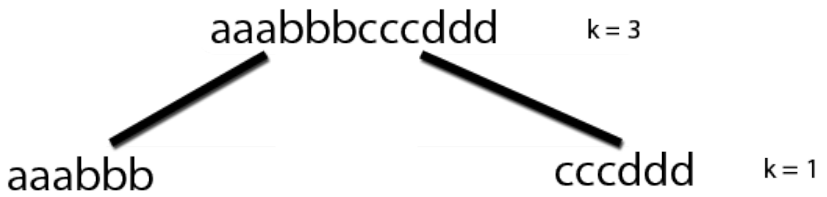
# Hierarchical approach

- divide the pattern into $k+1$ pieces hierarchically
  1. split the pattern in two pieces and search for each piece with $k = \lfloor \frac{k}{2} \rfloor$
  2. halves are recursively split and searched until error rate reaches zero



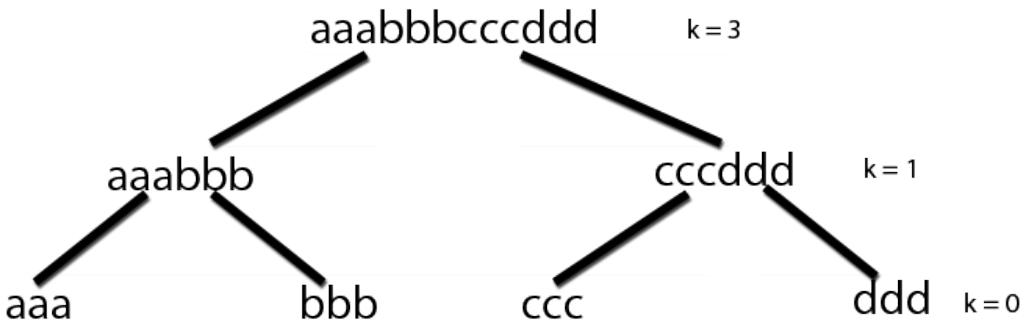
# Hierarchical approach

- divide the pattern into $k+1$ pieces hierarchically
  1. split the pattern in two pieces and search for each piece with $k = \lfloor \frac{k}{2} \rfloor$
  2. halves are recursively split and searched until error rate reaches zero



# Hierarchical approach

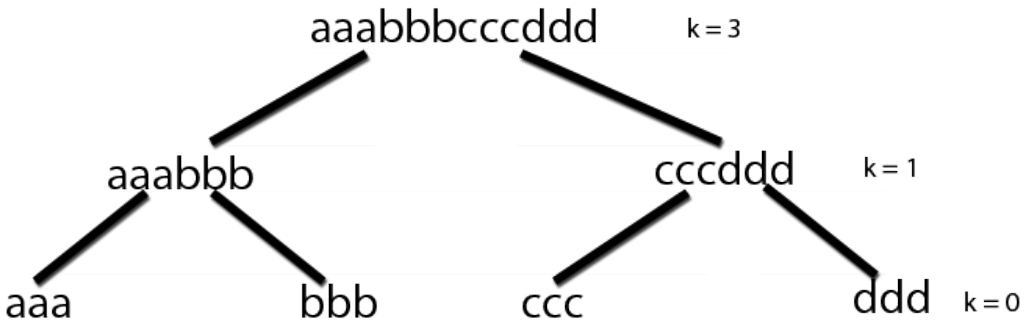
- divide the pattern into $k+1$ pieces hierarchically
  1. split the pattern in two pieces and search for each piece with $k = \lfloor \frac{k}{2} \rfloor$
  2. halves are recursively split and searched until error rate reaches zero



# Hierarchical approach - example

pattern: aaabbbcccddd

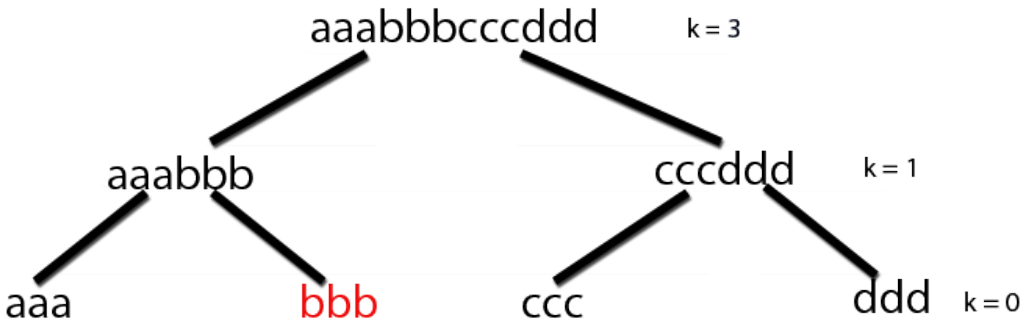
text: xxxbbbxxxxxx



# Hierarchical approach - example

pattern: aaabbbcccddd

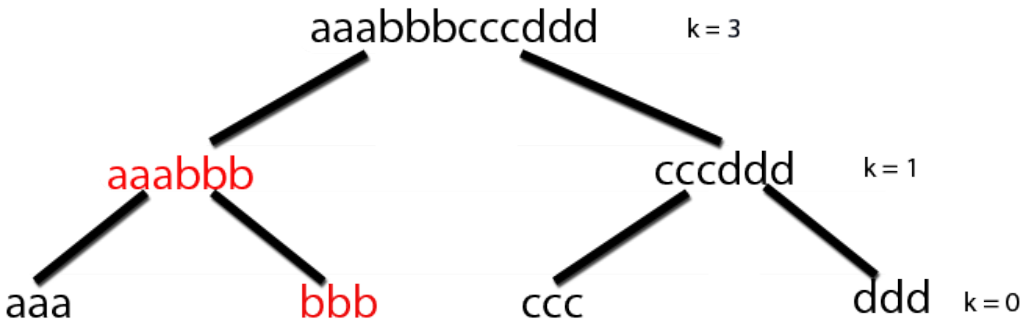
text: xxxbbbxxxxxx



# Hierarchical approach - example

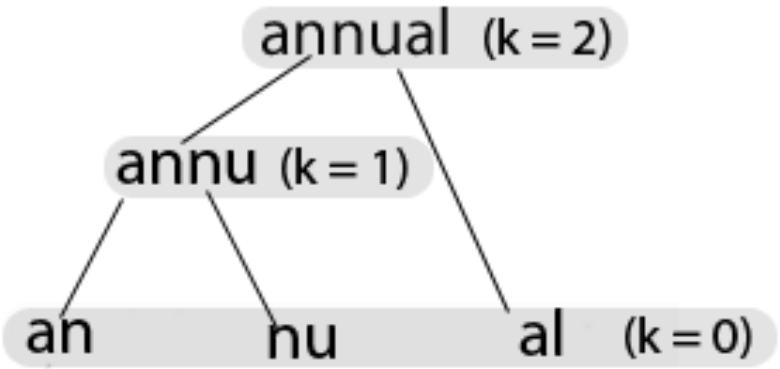
pattern: aaabbbcccddd

text: xxxbbbxxxxxx



# Hierarchical approach

**if** $k+1$ **not power of 2:**
as balanced as possible




*left*: number of pattern pieces in the left tree ($\lceil \frac{(k+1)}{(2)} \rceil$)

*lk*: $\lfloor \frac{(left \cdot k)}{(k+1)} \rfloor$

*right*: number of pattern pieces in the right tree ($k + 1 - left$)

*rk*: $\lfloor \frac{(right \cdot k)}{(k+1)} \rfloor$

# Outline

# QUASAR

"**Q[u]**-gram **A**lignment based on **S**uffix **AR**rays"

$\rightarrow$ computes approximate local matches

What's that?

Given two sequences Q(uery) and D(atabase), the pair of substrings Q' and D' with window length $w$ is *locally similar*

$\Leftrightarrow$

edit distance of Q' and D' $\leq k$

# QUASAR

"**Q[u]**-gram **A**lignment based on **S**uffix **AR**rays"

$\rightarrow$ computes approximate local matches

## What's that?

Given two sequences Q(uery) and D(atabase), the pair of substrings Q' and D' with window length *w* is *locally similar*

$\Leftrightarrow$

edit distance of Q' and D' $\leq k$

# QUASAR

"**Q[u]**-gram **A**lignment based on **S**uffix **AR**rays"

$\rightarrow$ computes approximate local matches

Important:

In Read-Mapping we only deal with the special case of semi-global alignments, so Q' is the entire read, w the read length.

# Q-Gram Lemma

**q-gram** a short sub-sequence of length q (a.k.a. k-mer)

**R** a read of length w

**D'** a region of length w in the Database with distance $\leq k$ to R

**Lemma** R and D' share

$$\geq t = w + 1 - (k + 1)q$$

common q-grams

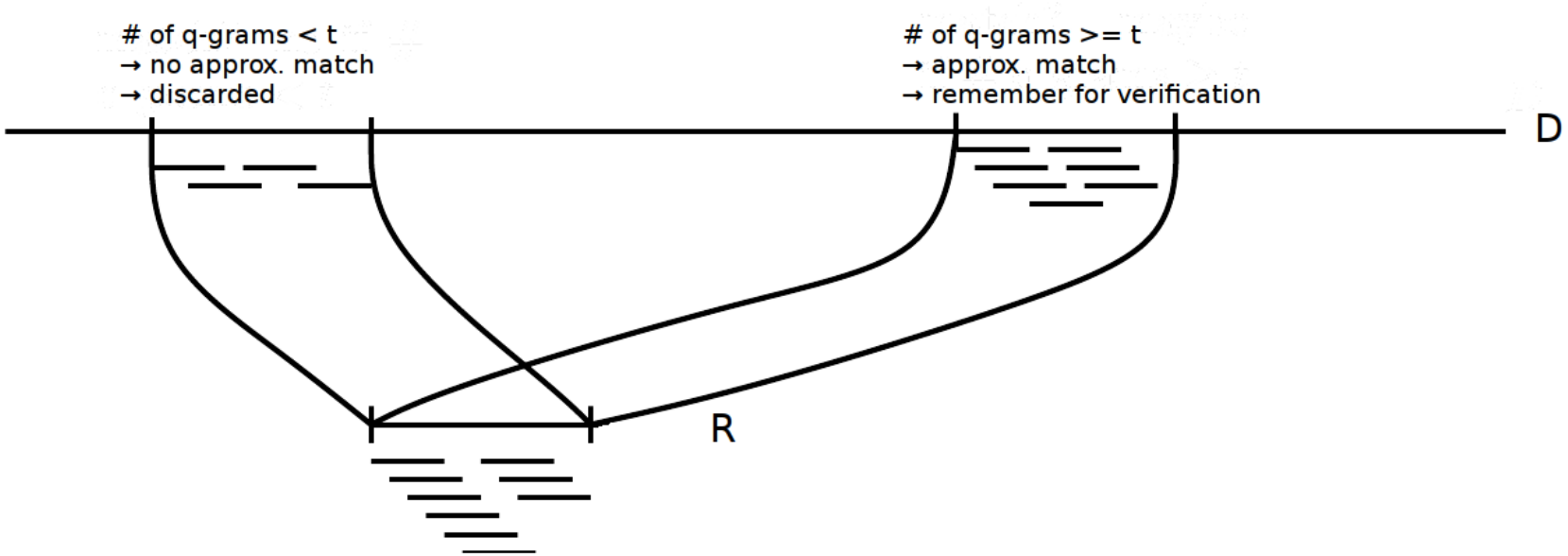
# Q-Gram Lemma

$\longrightarrow$ You will prove this lemma in the exercise!

## Hints

- How many common q-grams are there between two perfect matches?
- How many q-grams does a single mismatch destroy?

# Q-Gram Lemma


# of q-grams < t
→ no approx. match
→ discarded
# of q-grams >= t
→ approx. match
→ remember for verification
D
R

# QUASAR Algorithm

**Index** create a q-gram-index for O(1)-lookups

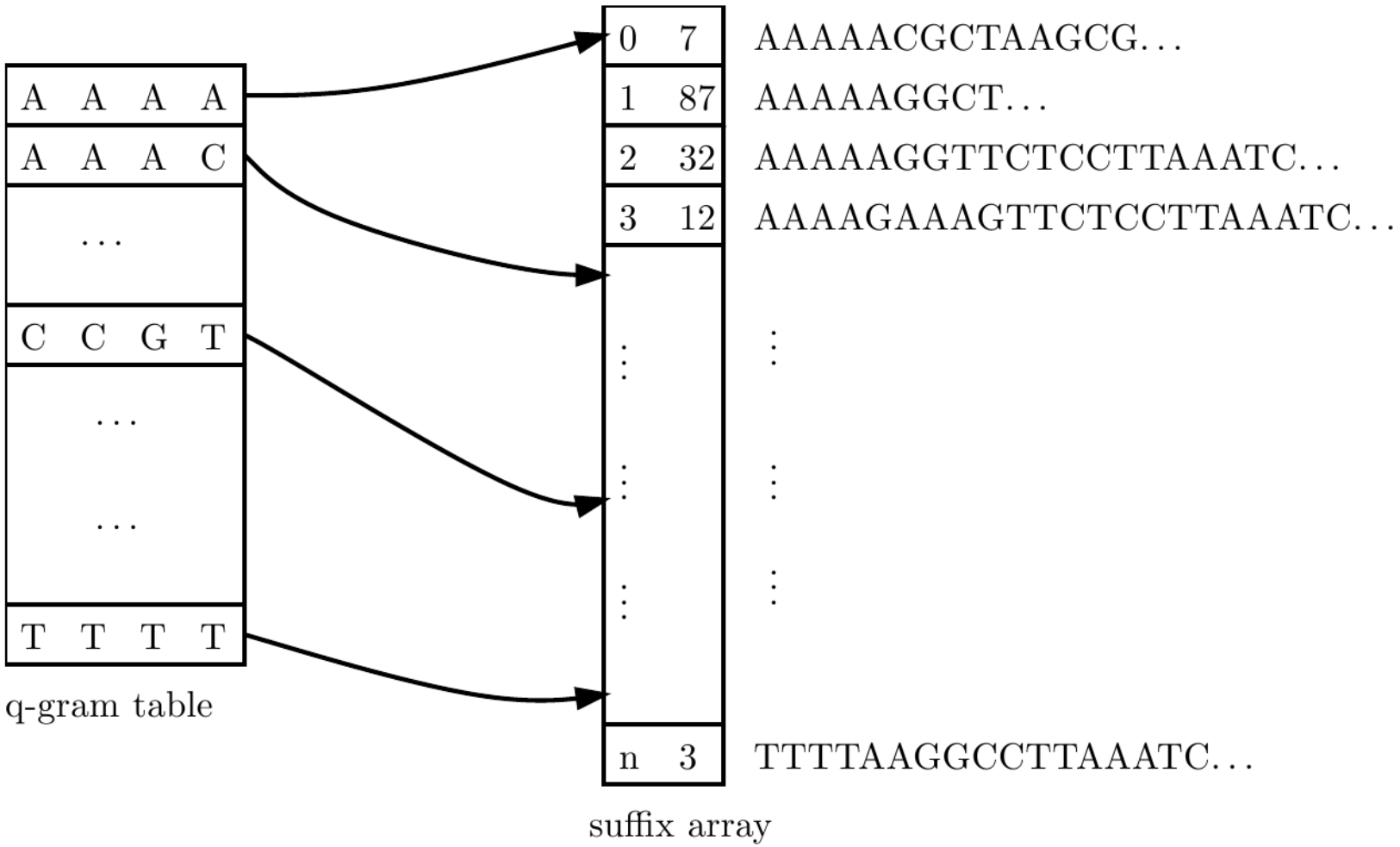
**Blocks** divide genome into non-overlapping blocks (buckets)

**Counting** lookup all q-grams in R and increment corresponding blocks' counters

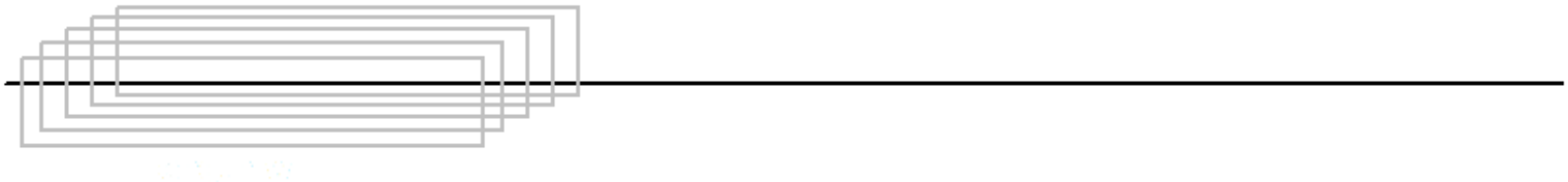
**Threshold** all blocks with counter $\geq t$ are remembered for verification

The last two steps are repeated for every read.

# Q-Gram-Index


A A A A
A A A C
...
C C G T
...
...
T T T T
q-gram table
0 7
1 87
2 32
3 12
n 3
AAAAACGCTAAGCG...
AAAAAGGCT...
AAAAAGGTTCTCCTTAAATC...
AAAAGAAAGTTCTCCTTAAATC...
TTTTAAGGCCTTAAATC...
suffix array

# Blocking

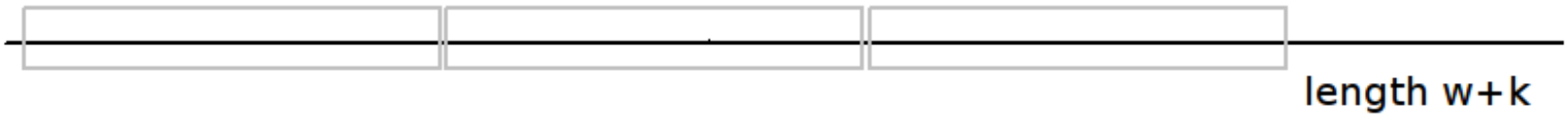


A trivial approach to counting would be to look at all (overlapping) sub-strings of length w ($w + k$ in case we allow gaps) in D and count the amount of matching q-grams. This would however require

$$|D| - w + 1 \textit{ counters}$$

$\rightarrow$ very memory expensive.

# Blocking




Alternatively you can devide the genome into *non-overlapping* buckets, called blocks and keep one counter for each. To not miss out on the approx. matches spanning a block border we introduce a second row of shifted blocks and increase the length to $\geq 2 * (w + k)$.

$$\rightarrow \frac{|D|}{w + k} \textit{ counters}$$

# Blocking


Alternatively you can devide the genome into *non-overlapping* buckets, called blocks and keep one counter for each. To not miss out on the approx. matches spanning a block border we introduce a second row of shifted blocks and increase the length to $\geq 2 * (w + k)$.

$$\rightarrow \frac{|D|}{w + k} \textit{ counters}$$

# Blocking

But keep in mind:

- because blocks are $\geq w$ long, reaching the threshold is only a necessary condition for containing an approx. match, it is not sufficient!
- $\rightarrow$ a larger block implies worse specificity

# QUASAR Algorithm

**Index** create a q-gram-index for O(1)-lookups

**Blocks** divide genome into non-overlapping blocks (buckets)

**Counting** lookup all q-grams in R and increment corresponding blocks' counters

**Threshold** all blocks with counter $\geq t$ are remembered for verification

The last two steps are repeated for every read.

# Outline

# Motivation

- the higher $q$ is, the less hits we have
  $\rightarrow$ higher filtration rate
- but the threshold also decreases ($t = w - q - qk + 1$)
  $\rightarrow$ lower efficiency
- solution: gapped q-grams result in a higher threshold

# Definition (non-formal)

| | | Example: ##.#..# |
|---|---|---|
| shape Q | set of $\mathbb{N}_0$ | $\{0, 1, 3, 6\}$ |
| size of Q: $\lvert Q \rvert$ | "number" of # | 4 |
| span of Q: $s(Q)$ | "number" of #, . | 7 |

only positions with # are checked, positions with . are ignored

# Thresholds

The q-gram lemma can be generalised to gapped q-grams:

$$t = w - s(Q) - |Q|k + 1$$

However it is not tight anymore...

# Thresholds

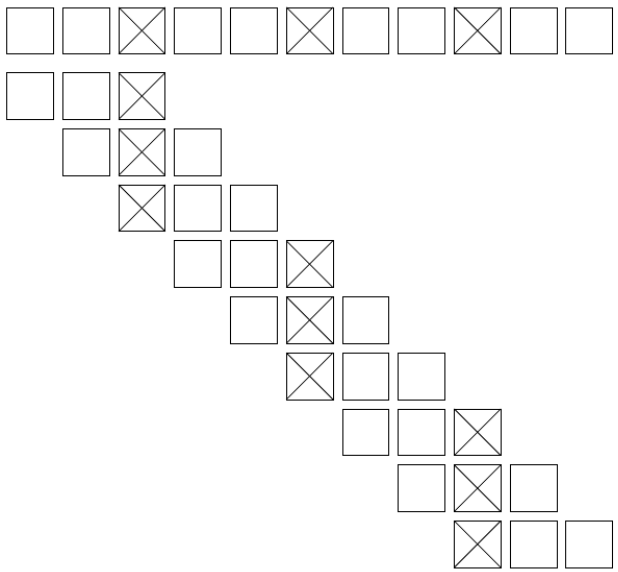
| shape | ### | ##.# |
|---|---|---|
| $t =$ | $11 - 3 - 3 * 3 + 1 = 0$ | $11 - 4 - 3 * 3 + 1 = -1$ |

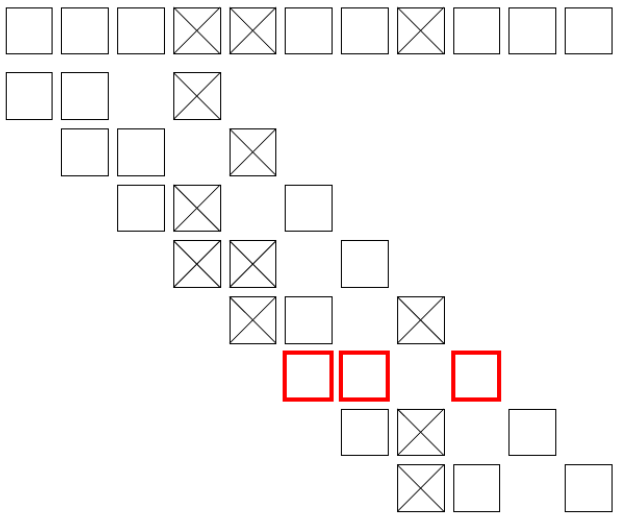
$w = 11$
$k = 3$

# Thresholds

| shape | ### | ##.# |
|---|---|---|
| $t =$ | $11 - 3 - 3 * 3 + 1 = 0$ | 1 |

$w = 11$
$k = 3$

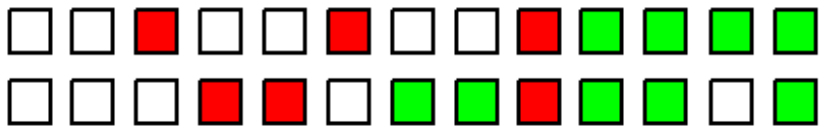




# Minimum Coverage

There is another intricacy when using gapped q-grams. Consider the shapes ### and ##.# for $w = 13$ and $k = 3$.

In both cases $t = 2$, but for ### four (consecutive) matches suffice, while ##.# requires five matches.



Obviously a higher minimum coverage increases the filter specificity.

# Summary

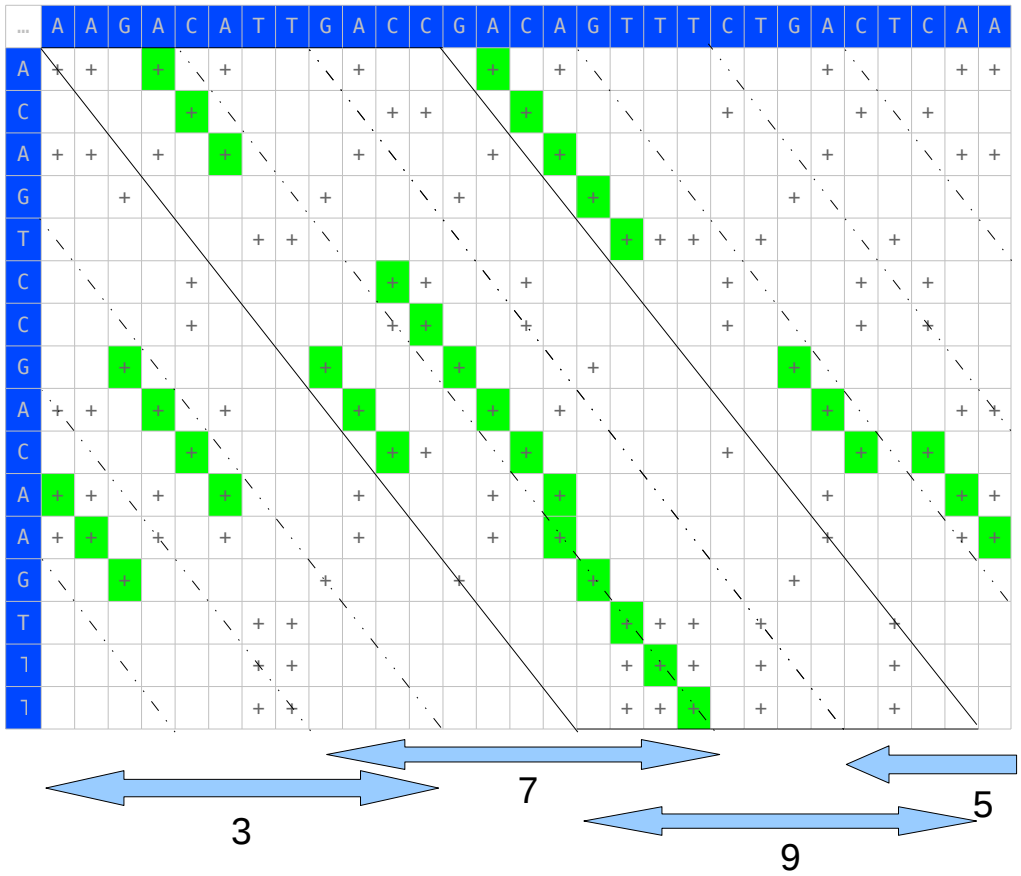
- Gapped Q-grams improve the filter efficiency by magnitudes
- placement of gaps in the Q-gram influences threshold and minimum coverage
- threshold and minimum coverage both influence filter efficiency
- there is no closed formula for computing the threshold of gapped Q-grams (the Q-Gram Lemma is only a lower bound)

# Outline

# Basic Idea

- if we do not allow gaps, then all q-grams are on one diagonal (trivial)
- if we do allow gaps, then the alignment spans at most $k + 1$ diagonals
  $\longrightarrow$ it is unnecessary to look outside of this parallelogram

# Basic Idea



# Basic Idea

- similar to the blocking described earlier, we do not count for every possible parallelogram of width $k+1$, but in wider overlapping parallelograms
- if the actual width $-k$ is a power of two the parallelograms can be computed by bit-shifting operations
- depending on implementation, parallelograms that reach the threshold are verified directly
  $\longrightarrow$ we only need to have a few counters that we “recycle”

# Outline

# Overview

Verification, in approaches that use q-gram-counting, is completely independent of the filtering step. It can be performed after all candidate regions have been identified (QUASAR), or “on-the-fly” (RazerS).

The algorithms employed range from traditional heuristic approaches, like BLAST (QUASAR) to specialised DP-based algorithms like Myers Bitvector algorithm (RazerS). When using hamming distance, scoring is trivial (count mismatches along the diagonal).

# Myers Bitvector algorithm - principal ideas

## Remember classical DP

- Needleman-Wunsch for global alignments, Smith-Waterman for local alignments.
- semi-global by setting only first row (not first column) to 0
- both use $O(nm)$ space and run-time.
- by only remembering the last column and doing a backtrace later, we can reduce space-requirement to $O(m)$ (Hirschberg)

# Myers Bitvector algorithm - principal ideas

## Ukkonen's algorithm

- observation that each cell's value differs $\{-1, 0, +1\}$ from its neighbours'
- based on that you can quickly realise when a value will never become "good" again in a column (once it has reached $k + 1$ ) and stop there
- this results in something similar to a banded alignment
- run-time O(km)

# Myers Bitvector algorithm - principal ideas

## Myers Bitvector algorithm

- do not save absolute values in the DP, but the differences to above cell ($\in \{-1, 0, +1\}$)

$\Delta v_{i,j}$:

| | | A | N | N | E | A | L | I | N | G |
|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| A | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| N | 1 | 1 | -1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 |
| N | 1 | 1 | 1 | -1 | -1 | 1 | 1 | 0 | 0 | 0 |
| U | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| A | 1 | 1 | 1 | 1 | 1 | -1 | -1 | 0 | 1 | 1 |
| L | 1 | 1 | 1 | 1 | 1 | 1 | -1 | -1 | -1 | 0 |

# Myers Bitvector algorithm - principal ideas

## Myers Bitvector algorithm

- do not save absolute values in the DP, but the differences to above cell ($\in \{-1, 0, +1\}$)
- columns then encoded as bit-vectors
- dependencies/relation of cells are encoded as bit-operations (AND, OR, OR NOT)
- columns are computed by bit-shifting similar to Shift-Or algorithm
- depending on read length a complete column maybe calculated simultaneously

# Thank you for your attention!

Questions?

Please, also read the exercise now and ask if you have problems understanding the tasks.