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

Introduction
   Review

Basic filtering approach
   Pigeonhole
   PEX

q-gram-counting
   QUASAR pipeline
   gapped q-grams
   filtering in parallelograms (SWIFT)
   verification algorithms
Outline

Introduction

Review

Basic filtering approach
- Pigeonhole
- PEX

q-gram-counting
- QUASAR pipeline
- gapped q-grams
- filtering in parallelograms (SWIFT)
- verification algorithms
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 $\Rightarrow$ 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 $\Rightarrow$ 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 $\implies$ 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:

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

Higher error level
- cost of verification increases
- reduction of filter efficiency

discard large segments of the text ⇒ fast search
⇒ improve average-case performance
filtering vs non-filtering

Filtering sensitive to error level:

\[ \alpha := \frac{k}{m} \quad (k = \text{error}; \ m = \text{pattern length}) \]

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:

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

Higher error level

- cost of verification increases
- reduction of filter efficiency

discard large segments of the text ⇒ fast search  
⇒ improve average-case performance
Outline

Introduction

Review

Basic filtering approach
Pigeonhole
PEX

q-gram-counting
QUASAR pipeline
gapped q-grams
filtering in parallelograms (SWIFT)
verification algorithms

RNA-Seq
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

Introduction
  Review

Basic filtering approach
  Pigeonhole
  PEX

q-gram-counting
  QUASAR pipeline
  gapped q-grams
  filtering in parallelograms (SWIFT)
  verification algorithms
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 $\text{Occ}$ 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 $i \in \{1, \ldots, j\}$, $\text{Occ}$ includes a substring that matches $p^i$ with $\lfloor \frac{a_i k}{A} \rfloor$ errors.
Introduction

Basic filtering approach

q-gram-counting

PEX

Basic procedure

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

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

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

RNA-Seq
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
Introduction

Basic filtering approach

q-gram-counting

PEX

Verification

- $p^i[start : end]$ 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

- $p^i[\text{start} : \text{end}]$ matches at $t[j : j + (\text{end}_i - \text{start}_i)]$
- occurrences are of length at most $m + k$
  - occurrence can start at most $\text{start}_i - 1 + k$ before position $j$ in $t$
  - can finish at most $m - \text{end}_i + k$ after $t[j + (\text{end}_i - \text{start}_i)]$

⇒ check the text area of
$t[j - (\text{start}_i - 1) - k : j + (m - \text{start}_i) + k] \Rightarrow$ length of text area $m + 2k$
Verification

- $p^i[start : end]$ 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 $t[j + (end_i - start_i)]$

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

- \( p^i[start : end] \) 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

- $p^i[start : end]$ 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
$t[j - (start_i - 1) - k : j + (m - start_i) + k] \Rightarrow \text{length of text area } m + 2k$
Introduction

Basic filtering approach

p^i[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)]

⇒ check the text area of
  t[j - (start_i - 1) - k : j + (m - start_i) + k] ⇒ length of text area m + 2k
An example

pattern: annual

text (t): any_annealing

error (k): 2
An example

\[ t = \text{any\_annealing} \]

**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 \(\text{None}\)
- al in \(t\) \(\Rightarrow\) pos 9

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

<table>
<thead>
<tr>
<th>(t[9])</th>
<th>(t[10])</th>
<th>(t[11])</th>
</tr>
</thead>
<tbody>
<tr>
<td>annea-</td>
<td>anneal</td>
<td>anneali</td>
</tr>
<tr>
<td>annual</td>
<td>annual</td>
<td>annual-</td>
</tr>
</tbody>
</table>
An example

\[ t = \text{any\_annealing} \]

**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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>annea-</td>
<td>anneal</td>
<td>anneali</td>
</tr>
<tr>
<td>annual</td>
<td>annual</td>
<td>annual</td>
</tr>
</tbody>
</table>

RNA-Seq
An example

\[ t = \text{any\_annealing} \]

**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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>anneal</td>
<td>anneal</td>
<td>anneali</td>
</tr>
<tr>
<td>annual</td>
<td>annual</td>
<td>annual</td>
</tr>
</tbody>
</table>
An example

\[ t = \text{any\_anneal\_ing} \]

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 \text{None}
- \( al \) in \( t \) \( \Rightarrow \) pos 9

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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>anneal</td>
<td>anneal</td>
<td>anneal_in</td>
</tr>
<tr>
<td></td>
<td>annual</td>
<td>annual</td>
<td>annual__</td>
</tr>
</tbody>
</table>

RNA-Seq
An example

\( t = \text{any\_annealing} \)

**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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>annea-</td>
<td>anneal</td>
<td>anneali</td>
</tr>
<tr>
<td></td>
<td>annual</td>
<td>annual</td>
<td>annual-</td>
</tr>
</tbody>
</table>

RNA-Seq
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

```
aaabbcccdde
```

$k = 3$
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 $k = \lceil \frac{k}{2} \rceil$
  2. halves are recursively split and searched until error rate reaches zero

```
   aaabbbcccddd   k = 3
  /\             /\  
 aaabbb         cccddd   k = 1
```

RNA-Seq
Hierarchical approach

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

---

![Diagram showing hierarchical approach]

- RNA-Seq
Hierarchical approach - example

- **pattern**: aaabbbccccddd
- **text**: xxxbbbxxxxxxx

The diagram illustrates the hierarchical approach with the pattern and text as shown. Each node represents a substring of the pattern or text, with a depth indicating the number of steps in the hierarchical filtering process.
Hierarchical approach - example

pattern: aaabbbccccddd
text: xxxbbbxxxxxx
Hierarchical approach - example

**pattern:** aaabbbccccddd

**text:** xxxbbbxxxxxx
Hierarchical approach

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

left: number of
pattern pieces in
the left tree
($\left\lfloor \frac{(k+1)}{2} \right\rfloor$)

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

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

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

Introduction
  Review

Basic filtering approach
  Pigeonhole
  PEX

q-gram-counting
  QUASAR pipeline
  gapped q-grams
  filtering in parallelograms (SWIFT)
  verification algorithms
QUASAR

“Q[u]-gram Alignment based on Suffix ARrays”

→ 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

⇔

edit distance of Q’ and D’ ≤ k
Introduction

Basic filtering approach

q-gram-counting

QUASAR pipeline

QUASAR

“Q[u]-gram Alignment based on Suffix ARrays”

→ computes approximate local matches

What’s that?
Given two sequences Q(query) and D(database), the pair of substrings Q’ and D’ with window length $w$ is *locally similar*

$\iff$

edit distance of Q’ and D’ $\leq k$
“Q[u]-gram Alignment based on Suffix ARrays”

→ 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

\[ t = w + 1 - (k + 1)q \]

common q-grams
Q-Gram Lemma

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
INTRODUCTION

Basic filtering approach

q-gram-counting

QUASAR pipeline

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 diagram illustrating the q-gram table and suffix array for RNA-Seq reads. The q-gram table contains q-grams such as `AAAAA`, `AAC`, `CAA`, and `TTT`, with corresponding suffix array positions and read segments. The diagram shows how these q-grams can be used to index and filter RNA-Seq reads efficiently.
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

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

→ very memory expensive.
Blocking

Alternatively you can divide 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 $\geq 2 \times (w + k)$.

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

Alternatively you can divide 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 $\geq 2 \times (w + k)$.

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

▶ → 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.
Introduction

Basic filtering approach
- q-gram-counting

Outline

Introduction
- Review

Basic filtering approach
- Pigeonhole
- PEX

q-gram-counting
- QUASAR pipeline

gapped q-grams
- filtering in parallelograms (SWIFT)
- verification algorithms
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)

<table>
<thead>
<tr>
<th>shape Q</th>
<th>set of $\mathbb{N}_0$</th>
<th>Example:</th>
</tr>
</thead>
<tbody>
<tr>
<td>size of Q: $</td>
<td>Q</td>
<td>$</td>
</tr>
<tr>
<td>span of Q: $s(Q)$</td>
<td>“number” of #, .</td>
<td>4</td>
</tr>
</tbody>
</table>

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>only positions with # are checked, positions with . are ignored</td>
</tr>
</tbody>
</table>

Example: 
```
##.#..#
```

RNA-Seq
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

<table>
<thead>
<tr>
<th>shape</th>
<th>###</th>
<th>#.#</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t =$</td>
<td>$11 - 3 - 3 \times 3 + 1 = 0$</td>
<td>$11 - 4 - 3 \times 3 + 1 = -1$</td>
</tr>
</tbody>
</table>

$w = 11$

$k = 3$
### Thresholds

<table>
<thead>
<tr>
<th>shape</th>
<th>###</th>
<th>#.#</th>
<th>$w = 11$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t =$</td>
<td>$11 - 3 - 3 \times 3 + 1 = 0$</td>
<td>$1$</td>
<td>$k = 3$</td>
</tr>
</tbody>
</table>

![Diagram of gapped q-grams]
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

Introduction
Review

Basic filtering approach
Pigeonhole
PEX

q-gram-counting
QUASAR pipeline
gapped q-grams
filtering in parallelograms (SWIFT)
verification algorithms
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
  \( \rightarrow \) it is unnecessary to look outside of this parallelogram
Introduction

Basic filtering approach

q-gram-counting

filtering in parallelograms (SWIFT)

**Basic Idea**

|   | A | A | G | A | C | T | T | G | A | C | A | G | T | T | T | C | T | G | A | C | T | C | A | A |
| A | + | + |   | + |   | + |   | + |   | + |   | + |   | + |   | + |   | + |   | + |   | + |   | + |   |
| C | + |   | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + |
| A | + | + |   | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + |
| G | + |   | + | + |   | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + |
| T | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + |
| C | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + |
| T | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + |
| C | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + |
| G | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + |
| A | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + |
| C | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + |
| A | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + |
| G | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + |
| T | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + |
| T | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + |
| T | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + |

RNA-Seq
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
  \[\rightarrow\] we only need to have a few counters that we “recycle”
Outline

Introduction
  Review

Basic filtering approach
  Pigeonhole
  PEX

q-gram-counting
  QUASAR pipeline
  gapped q-grams
  filtering in parallelograms (SWIFT)

verification algorithms
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)
Introduction

Basic filtering approach

q-gram-counting

verification algorithms

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 \((\in \{-1, 0, +1\})\)

\[
\begin{array}{cccccccccc}
& A & N & N & E & A & L & I & N & G \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
A & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\
N & 1 & 1 & -1 & 0 & 1 & 1 & 0 & 1 & 1 \\
N & 1 & 1 & 1 & -1 & -1 & 1 & 1 & 0 & 0 \\
U & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\
A & 1 & 1 & 1 & 1 & -1 & -1 & 0 & 1 & 1 \\
L & 1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & 0 \\
\end{array}
\]
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.