6 Q-gram filters for $\varepsilon$-matches

This exposition was developed by Clemens Gröpl. It is based on:


6.1 Motivation

Comparison of large genomic sequences can be speeded up a lot if filtering techniques are applied. The key observation is that a local alignment of high sequence similarity must contain at least a few short exact matches.

The idea of using q-grams for fast filtering is not new. A q-gram is a substring of length $q$. Programs like BLAST use q-grams which occur in both sequences as seeds for a local alignment search.

It has also been observed that combining the idea of seeds with a combinatorial argumentation based on some form of the pigeon hole principle can be used to discard large parts of the input sequences from further consideration, because they cannot contain a good local alignment.

We can distinguish three kinds of algorithms.

When applied for finding highly similar regions, the classical exact algorithms (e. g. Smith-Waterman) will spend most of the time verifying that there is no match between a given pair of regions. The running times (typically the product of sequence lengths) are infeasible for genome size sequences.

Heuristics like BLAST typically employ a q-gram index to locate seeds and perform a verification for the candidate regions located in this way. However, BLAST might fail to recognize an existing match, unless the filtering parameters are set very stringent. Thus one has to trade off sensitivity against speed.

A filter is an algorithm that allows us to discard large parts of the input, but is guaranteed not to lose any significant match. The trade-off to be considered for filtering algorithms is thus only whether the additional effort is payed off by the saving of time spent for verifications.

In this lecture, we will consider the problem of finding matches of low error rate $\varepsilon$ and a given minimum length $n_0$.

The cost measure will be the edit distance (Levenshtein distance). That is, the distance between two strings is the number of insertions, deletions, and substitutions needed to transform one into the other.

The edit distance can be computed by the well-known Needleman-Wunsch algorithm. It computes in $O(|A||B|)$ time an edit matrix $E(i,j) := \text{dist}(A[1..i], B[1..j])$. The letter $A[i]$ corresponds to the step from row $i-1$ to $i$, so it is natural to visualize the letters between the rows and columns of the edit matrix, etc..

An $\varepsilon$-match is a local alignment for substrings ($\alpha, \beta$) with an error rate of at most $\varepsilon$. That is, $\text{dist}(\alpha, \beta) \leq \varepsilon |\beta|$. (Note the ‘asymmetry’ in the definition of error rate.)

The problem can now be formally stated as follows:
Given a target string $A$ and a query string $B$, a minimum match length $n_0$ and a maximum error rate $\varepsilon > 0$;

Find all $\varepsilon$-matches $(\alpha, \beta)$ where $\alpha$ and $\beta$ are substrings of $A$ and $B$, respectively, such that

1. $|\beta| \geq n_0$ and
2. $\text{dist}(\alpha, \beta) \leq \lfloor \varepsilon |\beta| \rfloor$.

### 6.3 $q$-gram filters for $\varepsilon$-matches

A $q$-hit is a pair $(i, j)$ of indices such that $A[i..i+q-1] = B[j..j+q-1]$.

The basic idea of the $q$-gram method is as follows:

1. Find (enumerate) all $q$-hits between the query and the target strings.
2. Identify regions (in the Cartesian product of the strings) that have “enough” hits.
3. Such candidate regions are then subjected to a closer examination.

The concrete methods differ in the shape and the size of the regions.

The following lemma relates $\varepsilon$-matches $(\alpha, \beta)$ to parallelograms of the edit matrix. For a moment, we assume that the length of $\beta$ is known, so that we can work with an absolute bound on the distance.

An $n \times e$ parallelogram of the edit matrix consists of entries from $n+1$ consecutive rows and $e+1$ consecutive diagonals.

**Lemma 1.** Let $\alpha$ and $\beta$ be substrings of $A$ and $B$, respectively, and assume that $|\beta| = n$ and $\text{dist}(\alpha, \beta) \leq e$. Then there exists an $n \times e$ parallelogram $P$ such that

1. $P$ contains at least $T(n, q, e) := (n+1) - q(e+1)$ $q$-hits,
2. the $B$-projection of the parallelogram is $p_B(P) = \beta$,
3. the $A$-projection $p_A(P)$ of the parallelogram is contained in $\alpha$.

The $A$- and $B$-projections are defined as illustrated below.

The $A$-projection $p_A(P)$ of a parallelogram $P$ is defined as the substring of $A$ between the last column of the first row of $P$ and the first column of the last row of $P$.

The $B$-projection $p_B(P)$ of a parallelogram $P$ is defined as the substring of $B$ between the first and the last row of $P$.

(Note: these figures are taken from the RECOMB and GCB version, which uses the transposed matrix of the JCB article.)
Clearly, a q-hit \((i, j)\) corresponds to \(q + 1\) consecutive entries of the edit matrix along the diagonal \(j - i\). A q-hit is contained in a parallelogram if its corresponding matrix entries are.

The proof of Lemma 1 is straightforward: Consider the path of an optimal alignment of \(a\) and \(b\). At each row except for the last \(q\) ones, we have a q-gram unless there is an edit operation among the next \(q\) edges. Each edit operation can ‘destroy’ at most \(q\) q-hits.

So the case where \(|b|\) is fixed was easy. Next we consider \(\varepsilon\)-matches for \(|b| \geq n_0\). The following lemma is the combinatorial foundation of the SWIFT algorithm.

**Lemma 2.** Let \(a\) and \(b\) be substrings of \(A\) and \(B\), respectively, and assume that \(|b| \geq n_0\) and \(\text{dist}(a, b) \leq \varepsilon|b|\). Let 
\[
U(n, q, \varepsilon) := T(n, q, \lceil \varepsilon n \rceil) = (n + 1) - q(\lceil \varepsilon n \rceil + 1)
\]
and assume that the q-gram size \(q\) and the threshold \(\tau\) have been chosen such that 
\[
q < \left\lceil \frac{1}{\varepsilon} \right\rceil \quad \text{and} \quad \tau \leq \min \left\{ U(n_0, q, \varepsilon), U(n_1, q, \varepsilon) \right\},
\]
where 
\[
n_1 := \left\lceil \left( \lceil \varepsilon n_0 \rceil + 1 \right) / \varepsilon \right\rceil.
\]
Then there exists a \(w \times e\) parallelogram \(P\) such that:

1. \(P\) contains at least \(\tau\) q-hits whose projections intersect \(a\) and \(b\),
2. \(w = (\tau - 1) + q(e + 1)\),
3. \(e = \left\lfloor \frac{2\tau + q - 3}{1/\varepsilon - q} \right\rfloor\),
4. if \(|b| \leq w\), then \(p_b(P)\) contains \(b\), otherwise \(b\) contains \(p_b(P)\).

The purpose of Lemma 2 is as follows. Given parameters \(\varepsilon\) and \(n_0\), we can choose suitable values for \(q, \tau, w,\) and \(e\) using Lemma 2. Then we enumerate all parallelograms \(P\) with enough hits according to these parameters. All relevant \(\varepsilon\)-matches can be found in these regions.

**Proof of Lemma 2.** The lemma is proven in three steps:

1. Assuming there is an \(\varepsilon\)-match \((a, b)\) of length \(|b| = n \geq n_0\), show that there are at least \(\tau\) q-hits in the surrounding \(n \times \lceil \varepsilon n \rceil\) parallelogram.
2. Argue that there is a \(w \times e\) parallelogram that contains at least \(\tau\) q-hits, where \(w\) and \(e\) do not depend on \(n \geq n_0\).
3. Determine the dimensions \(w\) and \(e\) of such a parallelogram.

\[\ldots\text{details omitted}\ldots\]

| Table 1. Filter Parameters for \(\varepsilon = 0.05\), by Lemma 2 and Corollary 1, Respectively |
|-----------------------------------------------|-----------------------------------------------|-----------------------------------------------|
| \(n_0\) | \(q = 7\) | \(q = 9\) | \(q = 11\) |
| \(w\)  | 37  | 64  | 128 | 39  | 68  | 136 | 40  | 71  | 133 |
| \(e\)  | 2   | 4   | 9   | 2   | 4   | 9   | 2   | 4   | 8   |
| \(\tau\) | 17  | 30  | 59  | 13  | 24  | 47  | 8   | 17  | 35  |
| \(q = 11\) | | | | | | | | | |
| \(n_0\) | 28  | 29  | 41  | 42  | 43  | 44  | 45  | 46  | 47  |
| \(w\)  | 39  | 40  | 52  | 53  | 54  | 55  | 67  | 68  | 69  |
| \(e\)  | 2   | 2   | 3   | 3   | 3   | 3   | 4   | 4   | 4   |
6.4 Algorithm

The SWIFT algorithm relies on the q-gram filter for \( \epsilon \)-matches of length \( n_0 \) or greater. Using the parameters obtained from Lemma 2, it searches for all \( w \times e \) parallelograms which contain a sufficient number of q-grams.

In the preprocessing step, we construct a q-gram index for the target sequence \( A \). The index consists of two tables:

1. The occurrence table is a concatenation of the lists \( L(G) := \{ i \mid A[i..i + q - 1] = G \} \) for all q-grams \( G \in \Sigma^q \) in \( A \).
2. The lookup table is an array indexed by the natural encoding of \( G \) to base \( |\Sigma| \), giving the start of each list in the occurrence table.

Once the q-gram index is built, the \( w \times e \) parallelograms containing \( \tau \) or more q-hits can be found using a simple sliding window algorithm.

The idea is to split the (fictitious) edit matrix into overlapping bins of \( e + 1 \) diagonals. For each bin we count the number of q-hits in the \( w \times e \) parallelogram that is the intersection of the diagonals of the corresponding bin and the rows of the sliding window \( W_j := B[j..j + w] \).

As the sliding window proceeds to \( W_{j+1} \), the bin counters are updated to reflect the changes due to the q-grams leaving and entering the window.

Whenever a bin counter reaches \( \tau \), the corresponding parallelogram is reported. Overlapping parallelograms can be merged on the fly.

The space requirement for the bins is reduced by searching for somewhat larger parallelograms of size \( w \times (e + \Delta) \). Then each bin counts for \( e + \Delta + 1 \) diagonals, and successive bins overlap by \( e \) diagonals. While this will lead to more verifications, it reduces the number of bins which have to be maintained. In practice, \( \Delta \) is set to a power of 2, and bin indices are computed with fast bit-operations.
Each ‘candidate’ parallelogram must be checked for the presence of an $\varepsilon$-match. This can be done trivially by dynamic programming. Alternatively, one can use the knowledge about the $q$-grams in the $\varepsilon$-match to construct an alignment by sparse dynamic programming.

**Algorithm 2: Filter for identifying parallelograms for $\varepsilon$-matches**

**Input**: Query $B$; $q$-gram index $I$ for target $A$; parameters $w, \varepsilon, \tau$; and $\Delta = 2^w$

**Output**: Set of parallelograms $P$

1. Allocate and initialize array of bin records $Bins$
2. $P \leftarrow \emptyset$
3. for $j \leftarrow 0$ to $|B| - q$ do
4.   $G \leftarrow B[j \cdot j + q - 1]$
5.   $L(G) \leftarrow$ lookup occurrence list for $G$ in $I$
6.   foreach $i \in L(G)$ do
7.     $d \leftarrow |A| + j - i$
8.     $b_0 \leftarrow d \gg \text{bit } z$
9.     $b_m \leftarrow b_0 \mod |Bins|$
10.    $P \leftarrow P \cup \text{UpdateBin}(Bins[b_m], j, b_0 \ll \text{bit } z)$
11. if $(d \& \text{bit } (\Delta - 1)) < \varepsilon$ then
12.     $b_m \leftarrow (b_m + |Bins| - 1) \mod |Bins|$
13.     $P \leftarrow P \cup \text{UpdateBin}(Bins[b_m], j, (b_0 - 1) \ll \text{bit } z)$
14. if $(j - e) \mod (\Delta - 1) = 0$ then
15.     $b_0 \leftarrow (j - e) \gg \text{bit } z$
16.     $b_m \leftarrow b_0 \mod |Bins|$
17.     /* CheckAndResetBin is similar to lines 3–8 of UpdateBin */
18.     $P \leftarrow P \cup \text{CheckAndResetBin}(Bins[b_m], j, b_0 \ll \text{bit } z)$
19. $P \leftarrow P \cup \{ \text{remaining parallelograms in } Bins \}$
6.5 Results

Table 3. Filtration Ratios and Times for EST All-Against-All Comparison

<table>
<thead>
<tr>
<th>((\epsilon, n_0))</th>
<th>SWIFT</th>
<th>QUASAR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Filtration</td>
<td>Filtration, best ratio</td>
</tr>
<tr>
<td>(0.05, 50)</td>
<td>6.5 \times 10^{-6}</td>
<td>4.5 \times 10^{-4}</td>
</tr>
<tr>
<td>(0.04, 30)</td>
<td>4.5 \times 10^{-6}</td>
<td>4.0 \times 10^{-4}</td>
</tr>
<tr>
<td>(0.05, 30)</td>
<td>5.4 \times 10^{-6}</td>
<td>4.3 \times 10^{-4}</td>
</tr>
</tbody>
</table>

Table 2. Running Times for EST All-Against-All Comparison\(^a\)

<table>
<thead>
<tr>
<th>((\epsilon, n_0))</th>
<th>SWIFT</th>
<th>BLAST</th>
<th>SSEARCH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running time</td>
<td>18 s</td>
<td>29 s</td>
<td>35 s</td>
</tr>
</tbody>
</table>

\(^a\)The time for the database formatting and preprocessing in BLAST (3 s) and SWIFT (12 s) is not included.
7 Applications of 2nd-Gen Sequencing

This exposition was developed by David Weese. It is based on:


7.1 Second-generation sequencing technologies

<table>
<thead>
<tr>
<th></th>
<th>454 FLX/Roche</th>
<th>Solexa/Illumina</th>
<th>SOLiD/ABI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequencing approach</td>
<td>pyrophosphate release</td>
<td>bridge amplification</td>
<td>ligation</td>
</tr>
<tr>
<td>Read lengths</td>
<td>400–500bp</td>
<td>36bp</td>
<td>35bp or 25bp (MP)</td>
</tr>
<tr>
<td>Mate pairs</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Output/Run</td>
<td>400–600Mbp in 10h</td>
<td>&gt; 1.5Gbp in 2.5d</td>
<td>3–4Gbp in 6d</td>
</tr>
<tr>
<td>Accuracy depends on</td>
<td>homopolymer length (&gt; 6 problematic)</td>
<td>nucleotide position in the read</td>
<td>nucleotide position in the read</td>
</tr>
</tbody>
</table>

GS FLX Titanium Series  
Genome Analyzer 2  
SOLiD System 2.0 Analyzer

7.2 Second-generation sequencing applications

- **Genome Assembly**
  - Resequesting
  - De novo assembly

- **RNA-Sequencing**
  - Expression Profiling
  - Alt. splicing  
  - microRNA

- **Genome Comparison**
  - SNPs
  - Rearrangements

- **Metagenomics**

- **Epigenetics**
  - ChIP-Seq
  - DNA Methylation
7.3 Motivation

Fundamental to almost all of these applications is the following problem:

**Problem 1 (Read Mapping Problem).** Given a set of read sequences $\mathcal{R}$, a reference sequence $G$, and a distance $k \in \mathbb{N}$. Find all pairs $(r, g)$ with $r \in \mathcal{R}$, $g$ is substring of $G$ and $\text{dist}(r, g) \leq k$.

Common distance measures are Hamming distance or edit distance. The pairs $(r, g)$ are called matches of $r$.

(Some of the) Existing Read Mapping Tools:

<table>
<thead>
<tr>
<th>Filtering Technique</th>
<th>Eland</th>
<th>Maq</th>
<th>SOAP</th>
<th>Seqmap</th>
<th>Zoom</th>
<th>Shrimp</th>
<th>RazerS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distance measure in filtering step</td>
<td>Hamming</td>
<td>Hamming</td>
<td>Hamming</td>
<td>both</td>
<td>Hamming</td>
<td>both</td>
<td>both</td>
</tr>
<tr>
<td>Distance measure in mapping step</td>
<td>Hamming</td>
<td>Hamming (Smith-Wat. for second mate)</td>
<td>Hamming (optionally with one gap)</td>
<td>Hamming or edit with at most 5 errors</td>
<td>Hamming or edit with at most one gap</td>
<td>Smith-Waterman</td>
<td>Either edit or Hamming</td>
</tr>
<tr>
<td>Supported read length</td>
<td>$\leq 32$</td>
<td>$\leq 127$</td>
<td>$\leq 60$</td>
<td>arbitrary</td>
<td>$\leq 63$</td>
<td>arbitrary</td>
<td>arbitrary</td>
</tr>
<tr>
<td>Sensitivity</td>
<td>full sensitivity only for up to 2 errors</td>
<td>full sensitivity</td>
<td>depends on setting, no switch to guarantee full sensitivity</td>
<td>full sensitivity</td>
<td>switch to guarantee full sensitivity</td>
<td>no help for parameter choice, default will be lossy for most settings</td>
<td>arbitrarily adjustable</td>
</tr>
<tr>
<td>Can output all (suboptimal) hits</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Observation 2** (Sharpness). The $q$-gram Lemma is sharp. The worst case occurs if the $k$ errors are equidistantly distributed.

The error probability increases with the nucleotide position for Illumina, SOLiD, and Sanger sequencing.

**Observation 3.** The worst case occurrence probability is very small.

Why not increasing $t$ or $q$ to increase filtration specificity and reduce runtime?

How many matches would be lost?

**Definition 4. Sensitivity** is the probability that a true match is classified as potential match:

$$P(\#\text{matching } q\text{-grams } \geq t \mid \#\text{errors } \leq k)$$

Loss Rate $= 1 - \text{Sensitivity}$
7.4 How to calculate the sensitivity

Let \( p^*_i \) be the probability of a sequencing error at nucleotide position \( i \).

We could enumerate all configurations of \( e = 0, \ldots, k \) errors and sum up the occurrence probs of those with \( \geq t \) matching \( q \)-grams.

\[
\text{genome} \quad \text{ACAAAGTCCGACAAGTTT} \\
\text{read} \quad \text{ACAAATGACCGACACGTTT} \quad \text{match with 3 replacements}
\]

The occurrence probability would be \( p^*_1 \cdot p^*_2 \cdot p^*_3 \cdot p^*_4 \cdot p^*_5 \cdot p^*_6 \cdot p^*_7 \cdot p^*_8 \cdot p^*_9 \cdot p^*_10 \cdot p^*_11 \cdot p^*_12 \cdot p^*_13 \cdot p^*_14 \cdot p^*_15 \cdot p^*_16 \cdot p^*_17 \cdot p^*_18 \), with \( p^*_i = 1 - p^*_i \).

The enumeration would take \( \Omega \left( \binom{n}{k} k \right) \) time. Not feasible for \( n = 200 \) and \( k = 20 \) errors.

7.5 Definitions

- Considering mismatches only (Hamming distance)

\[
\text{transcript} \quad T \in \Delta^*, \Delta = \{M, R\}, |T| = |\text{read}|
\]

- Considering mismatches and indels (edit distance)

\[
\text{transcript} \quad T \in \Delta^*, \Delta = \{M, R, I, D\}, \quad |\text{read}| \leq |T| \leq |\text{read}| + \#\text{errors}
\]

- Substrings \( M^q \) are called \( q \)-matches

- A \( q \)-match corresponds to a common \( q \)-gram

7.6 DP Approach (Hamming)

- Much more efficient than the full enumeration

- Recursive enumeration of all error configurations explicitly storing only the last \( q \) positions

- DP-Matrix \( R \) with \( R(i, e, t, T_2) \)

- \( i = 1 \)st transcript length
  - \( 0, \ldots, |\text{read}| - q \)

- \( e = \) number of errors
  - \( 0, \ldots, k \)

- \( t = \) threshold
  - \( 0, \ldots, r - q + 1 \)

- \( T_2 = \) 2nd transcript of length \( q \)
  - \( T_2 \in \Delta^q \) where \( \Delta = \{M, R\} \)

- Contains the sum of occurrence probabilities of transcripts \( T_1 \) s.t. \( T_1 \) contains \( e \) letters \( R \), \( T_1T_2 \) contains \( \geq t \) substrings \( M^q \)
The sum of occ. probs. of all transcripts $T_1$ ...

$T_1$ contains $\varepsilon$ errors
$T_1, T_2$ contains $\geq t$ substrings $M$

... can be computed recursively from

$\begin{align*}
  T'_1 & \quad X \quad T'_2 \\
  T'_1 & \text{contains } \begin{cases} 
    \varepsilon, & \text{if } x = M \\
    \varepsilon - 1, & \text{if } x = R 
  \end{cases} \text{ errors} \\
  T'_1 \times T'_2 & \text{contains } \begin{cases}
    t - 1, & \text{if } T_2 = M \\
    t, & \text{else} 
  \end{cases} \text{ substrings } M
\end{align*}$

Lemma 5 (Hamming distance, ungapped).

$R(0, \varepsilon, l, T_2) = \begin{cases} 
    1, & \text{if } \varepsilon = 0, l \leq \delta(T_2) \\
    0, & \text{else}
\end{cases}$

$R(i, \varepsilon, l, T_2) = p_M^\varepsilon \cdot R(i - 1, \varepsilon, l - \delta(T_2), MT_2[1..q - 1]) + p_R^\varepsilon \cdot R(i - 1, \varepsilon - 1, l - \delta(T_2), RT_2[1..q - 1])$

$\delta(T) := \begin{cases}
    1, & \text{if } T = M \\
    0, & \text{else}
\end{cases}$

We devised a DP algorithm that calculates the sensitivities for all $\varepsilon = 0, \ldots, k$ and $l = 1, \ldots, t_{\text{max}}$ in $O(n \cdot k \cdot t_{\text{max}} \cdot 2^q)$.

The recursion can be extended to gapped shapes and edit distance.

Example error distribution

Different loss rates of matches with 2 replacements
7.7 Filtration

Consider dotplots between genome and reads.

A match with $k$ indels is covered by at most $k + 1$ consecutive diagonals.
Cover the dot plots with parallelograms of \( w \) diagonals with \( w \geq k + 1 \) ...

... and an overlap of \( k \) diagonals.
Every possible sequence of \( k + 1 \) diagonals resides in a parallelogram.
Search common q-grams while scanning the genome from left ...

... to right.
Potential matches are contained in parallelograms with $\geq t$ common $q$-grams.

Optimizations as suggested in [RasStoMye05]:

- Associate every parallelogram with a counter
- Reuse counters after $(|r| - w - q)$ $q$-gram sliding steps
- Choose parallelogram width, s.t. they begin at multiples of a power of 2 → Fast bit-shift reveals counter number of a diagonal

Differences compared to Swift:

- not local, but semi-global alignment
- parallelograms are not opened or closed, they are verified as a whole

### 7.8 Verification

- **Edit Distance**
  - Parallelograms are verified with bit-vector algorithm by [Myers99]
  - It exploits hardware parallelism of bit-operations:
    - A 64-bit CPU calculates 64 DP cells in 14 arithmetic/logic operations
  - Returns the end position of a true match in the genome
  - Can be modified to also return the beginning

- **Hamming Distance**
  - Scan each diagonal until $k + 1$ mismatches occur
  - Every diagonal with $\leq k$ mismatches is a true match
7.9 Filtration Parameter Choice

Automatically choose filtering parameters (shape $Q$ and threshold $t$) to ... 

- achieve a certain sensitivity level
- minimize the running time of the mapping procedure

Therefore precompute the loss rates for ... 

- read lengths from 24 to 100
- error rates up to 10%
- a typical Illumina error profile [Dohm08]

Parameters for larger reads are extrapolated from precalculations with the same error rate.

Parameter tables can be recomputed with user-specific error distribution from:

- **Quality based probabilities**: Transform the average base call quality value for each position into a probability value.
- **A posteriori probabilities**: Map a small subset of reads and determine the position dependent error frequency.

For instance, parameters of 50bp reads can be recalculated within 10min using the DP algorithm.

7.10 Paired-End Read Mapping

Given a library size $\mu$ and a tolerated deviation $\delta$, we want to find all paired-end matches with

- an insertion size between $\mu - \delta$ and $\mu + \delta$,
- each mate matches with up to $k$ errors.

Paired-end reads are sequenced from different strands and “look at each other”.

There are two symmetries:
Search read 1 and the reverse-complemented read 2 on the same strand.

Scan one strand with two Swift filters from left . . .

. . . to right.
Keep their distance and record the potential matches of the trailing filter in a fifo.

Each true paired-end match is recognized as a Swift 2 potential match and a potential mate match in the fifo.

Implementation:
- As optimization use a last-seen-at table for Swift 1 potential matches.
- Output a true match with minimal errors and a minimal library deviation.

### 7.11 Results

The following datasets were used:

**Read sets**\(^1\) from the NCBI short read archive:
- 10,760,364 × 36bp reads of Drosophila melanogaster (SRR001815)
- 7,894,743 × 2 × 76bp reads of a human HapMap individual (SRR006387) trimmed to 63bp (Zoom’s limit)

**Reference genomes:**
- Drosophila melanogaster genome from FlyBase, Release 5.9
- Human genome from NCBI, Build 36.3

Verification of Expected Sensitivity:

\(^1\)In both sets, the Illumina technology was used.
1. Simulated reads
   - Simulate reads using a positional error model [Dohm08]
   - Group them according number of errors
   - **Empirical sensitivity** is the proportion that could be mapped back to origin

2. Real data
   - Map Drosophila reads lossless and keep those, that map uniquely
   - Group them according number of errors
   - Determine positional error model
   - **Empirical sensitivity** is the proportion that could be mapped back to origin

For different filtration settings \( q = 8 \ldots 14, t = 1 \ldots 20 \) we calculated the **estimated loss rate** and compared it with the **empirical loss rate**.

---

**Short Read Experiments:**

1. **Drosophila, Hamming distance 2**
   - Map Drosophila reads onto the Drosophila genome with \( \leq 2 \) mismatches
   - Shrimp uses Smith-Waterman verifier \( \rightarrow \) we adapted the scores (mismatch = 0, match = 1, gap penalties = -1000, score threshold = 34).

2. **Drosophila, edit distance 2**.
   - As in experiment (1) allowing also indels
   - Shrimp computes local alignments \( \rightarrow \) no scoring scheme for semi-global edit distance alignments
   - We emulated edit distance with (mismatch = -1, match = 1, gap penalties = -1, score threshold = 32)

3. **Human, Hamming distance 5**.
   - Map HapMap reads onto the human genome with \( \leq 5 \) mismatches
   - Adapted Shrimp scores as in experiment (1)
Paired-end read mapping onto unmasked human chromosome 21

- $2 \times 1,000,000$ and $2 \times 7,894,743$ reads of length 63
- up to 5 mismatches
- full and 99% sensitivity

Paired-end read mapping onto unmasked human chromosome 21

<table>
<thead>
<tr>
<th>experiment</th>
<th>RazerS100</th>
<th>RazerS99</th>
<th>Zoom</th>
<th>Shrimp</th>
<th>SeqMap</th>
<th>Soap</th>
<th>Maq</th>
</tr>
</thead>
<tbody>
<tr>
<td>all</td>
<td>time (min)</td>
<td>3.14</td>
<td>0.40</td>
<td>26.1</td>
<td>10.7</td>
<td>48.8</td>
<td>3.88</td>
</tr>
<tr>
<td></td>
<td>space (GB)</td>
<td>1.14</td>
<td>1.86</td>
<td>1.27</td>
<td>6.10</td>
<td>8.10</td>
<td>6.20</td>
</tr>
<tr>
<td></td>
<td>mapped pairs</td>
<td>352,725</td>
<td>351,767</td>
<td>352,617</td>
<td>352,742</td>
<td>349,721</td>
<td>354,020</td>
</tr>
</tbody>
</table>

Read mapping onto unmasked human chromosome 21

- 500,000 simulated 125bp and 250bp reads
- up to 8% errors, full and 99% sensitivity
7.12 Second-generation sequencing applications

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>RazerS100</th>
<th>RazerS99</th>
<th>Shrimp100</th>
<th>Shrimp99</th>
<th>Maq</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hamming</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>125bp time (min)</td>
<td>8.53</td>
<td><strong>4.15</strong></td>
<td>9.61 h</td>
<td>60.9</td>
<td>4.54</td>
</tr>
<tr>
<td>125bp space (GB)</td>
<td>0.80</td>
<td>1.54</td>
<td>1.93</td>
<td>4.48</td>
<td>0.38</td>
</tr>
<tr>
<td>mapped</td>
<td>500,000</td>
<td>499,991</td>
<td>500,000</td>
<td>499,991</td>
<td>405,377</td>
</tr>
<tr>
<td>250bp time (min)</td>
<td>14.7</td>
<td><strong>6.65</strong></td>
<td>32.9 h</td>
<td>160</td>
<td>-</td>
</tr>
<tr>
<td>250bp space (GB)</td>
<td>1.26</td>
<td>2.00</td>
<td>1.46</td>
<td>2.61</td>
<td>-</td>
</tr>
<tr>
<td>mapped</td>
<td>500,000</td>
<td>500,000</td>
<td>500,000</td>
<td>500,000</td>
<td>-</td>
</tr>
<tr>
<td><strong>edit</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>125bp time (min)</td>
<td>65.0</td>
<td><strong>23.7</strong></td>
<td>9.44 h</td>
<td>61.6</td>
<td>-</td>
</tr>
<tr>
<td>125bp space (GB)</td>
<td>0.74</td>
<td>1.71</td>
<td>0.84</td>
<td>4.50</td>
<td>-</td>
</tr>
<tr>
<td>mapped</td>
<td>500,000</td>
<td>500,000</td>
<td>500,000</td>
<td>500,000</td>
<td>-</td>
</tr>
<tr>
<td>250bp time (min)</td>
<td>55.8</td>
<td><strong>39.6</strong></td>
<td>14.7 h</td>
<td>28.5 h</td>
<td>-</td>
</tr>
<tr>
<td>250bp space (GB)</td>
<td>1.21</td>
<td>2.18</td>
<td>1.38</td>
<td>5.45</td>
<td>-</td>
</tr>
<tr>
<td>mapped</td>
<td>500,000</td>
<td>499,940</td>
<td>500,000</td>
<td>499,940</td>
<td>-</td>
</tr>
</tbody>
</table>

**Genome Assembly**
- Resequencing
- De novo assembly

**RNA-Sequencing**
- Expression Profiling
- Alt. splicing
- microRNA

**Genome Comparison**
- SNPs
- Rearrangements

**Metagenomics**
- ChIP-Seq
- DNA Methylation

**Epigenetics**
7.13 RNA-Sequencing

How RNA-Seq works:

- RNA isolation
- Reverse transcription to cDNA
- Fragmentation
- (Size selection)
- Sequencing

RNA-Seq applications:

- **Expression profiling:** Quantify gene expression levels
- **Alternative splicing:** Which mRNAs are generated from the same gene?
- **microRNA:** Where is the genomic source, which genes are regulated?

7.14 RNA-Seq - Alternative Splicing

Two approaches to determine splice variants:

1. Cut the genome at known splice sites and map mRNA reads onto combinations of merged genome fragments.
2. Map as many mRNA reads as possible onto the genome and use coverage and known introns to detect new splice sites. Proceed as above.
Second Approach*: 

- Map reads
- Assemble uniquely mapped reads
- Generate possible splices
- Try to map the non-uniquely mapped reads onto splices

---

*Tranell C, Pachter L, Salzberg SL. (2009) TopHat: discovering splice junctions with RNA-Seq, Bioinformatics

---

7.15 Genome Comparison

- Sequence paired-end reads of an unknown genome (sample)
- Map them onto a known reference genome (target)
- Search for small mutations (SNPs) or large structural variations (rearrangements) between them
A deletion in the sample induces pairs of reads to be farther apart than predicted.

Inversions, deletions, translocations can also be detected.\(^2\)


7.16 Other applications

ChIP-Sequencing

Metagenomics

7 Selected Publications

