Pattern Matching

- What if, instead of finding repeats in a genome, we want to find all sequences in a database that contain a given pattern?

- This leads us to a different problem, the **Pattern Matching Problem**.

Pattern Matching Problem

- Goal: Find all occurrences of a pattern in a text.

- Input:
  - Pattern $\boldsymbol{p} = p_1 \ldots p_n$ of length n
  - Text $\boldsymbol{t} = t_1 \ldots t_m$ of length m

- Output: All positions $1 \leq i \leq (m - n + 1)$ such that the n-letter substring of text $\boldsymbol{t}$ starting at i matches the pattern $\boldsymbol{p}$.

- **Motivation**: Searching database for a known pattern.

Exact Pattern Matching: A Brute Force Algorithm

PatternMatching(**p**,**t**)

1 $n \leftarrow$ length of pattern **p**

2 $m \leftarrow$ length of text **t**

3 for $i \leftarrow 1$ to $(m - n + 1)$

4 if $\boldsymbol{t_i \ldots t_{i+n-1}} = \mathbf{p}$

5 output i

Exact Pattern Matching: An Example

- *PatternMatching* algorithm for:
  - Pattern GCAT
  - Text AGCCGCATCT

Exact Pattern Matching: An Example

- *PatternMatching* algorithm for:
  - Pattern GCAT
  - Text AGCCGCATCT

- AGCCGCATCT
- GCAT

Exact Pattern Matching: An Example

- *PatternMatching* algorithm for:
  - Pattern GCAT
  - Text AGCCGCATCT

- AGCCGCATCT
- GCAT

Exact Pattern Matching: An Example

- *PatternMatching* algorithm for:
  - Pattern GCAT
  - Text AGCCGCATCT

- AGCCGCATCT
- GCAT

Exact Pattern Matching: An Example

- *PatternMatching* algorithm for:
  - Pattern GCAT
  - Text AGCCGCATCT

- AGCCGCATCT
- GCAT

Exact Pattern Matching: An Example

- *PatternMatching* algorithm for:
  - Pattern GCAT
  - Text AGCCGCATCT

- AGCCGCATCT
- GCAT

Exact Pattern Matching: An Example

- *PatternMatching* algorithm for:
  - Pattern GCAT
  - Text AGCCGCATCT

- AGCCGCATCT
- GCAT

Exact Pattern Matching: An Example

- *PatternMatching* algorithm for:
  - Pattern GCAT
  - Text AGCCGCATCT

- AGCCGCATCT
- GCAT

Exact Pattern Matching: Running Time

- *PatternMatching* runtime: O(*nm*)
  - In the worst case, we have to check *n* characters at each of the *m* letters of the text.
  - **Example**: Text = AAAAAAAAAAAAAAAAA, Pattern = AAAC

- This is rare; on average, the run time is more like O(*m*).
  - Rarely will there be close to *n* comparisons at each step.

- Better solution: **suffix trees**…solve in O(*m*) time.

- First we need **keyword trees**.

Approximate Pattern Matching Problem

- Goal: Find all approximate occurrences of a pattern in a text.

- Input: A pattern $\boldsymbol{p} = p_1 \ldots p_n$, text $\boldsymbol{t} = t_1 \ldots t_m$, and k, the maximum allowable number of mismatches.

- Output: All positions $1 \leq i \leq (m - n + 1)$ such that $t_i \ldots t_{i+n-1}$ and $p_1 \ldots p_n$ have at most k mismatches. i.e., HammingDistance($t_i \ldots t_{i+n-1}$, p) $\leq k$.

Approximate Pattern Matching: Brute Force

ApproximatePatternMatching(p, t, ***k*****)**

1. $n \leftarrow$ length of pattern **p**
2. $m \leftarrow$ length of text **t**
3. **for** $i \leftarrow 1$ to $m - n + 1$
4. $\quad dist \leftarrow 0$
5. $\quad$ **for** $j \leftarrow 1$ to n
6. $\quad\quad$ **if** $t_{i+j-1} \neq p_j$
7. $\quad\quad\quad dist \leftarrow dist + 1$
8. $\quad$ **if** $dist \leq k$
9. $\quad\quad$ **output** i

Approximate Pattern Matching: Running Time

- The brute force algorithm runs in O(*mn*) time.
- Landau-Vishkin algorithm: Gives improvement to O(*km*).
- We will next generalize the "Approximate Pattern Matching Problem" into a "Query Matching Problem."
  - Rather than patterns, we want to match *substrings* in a query to substrings in a text with at most ***k*** mismatches.
- **Motivation**: We want to see similarities to some gene, but we may not know which parts of the gene to look for.

Query Matching Problem

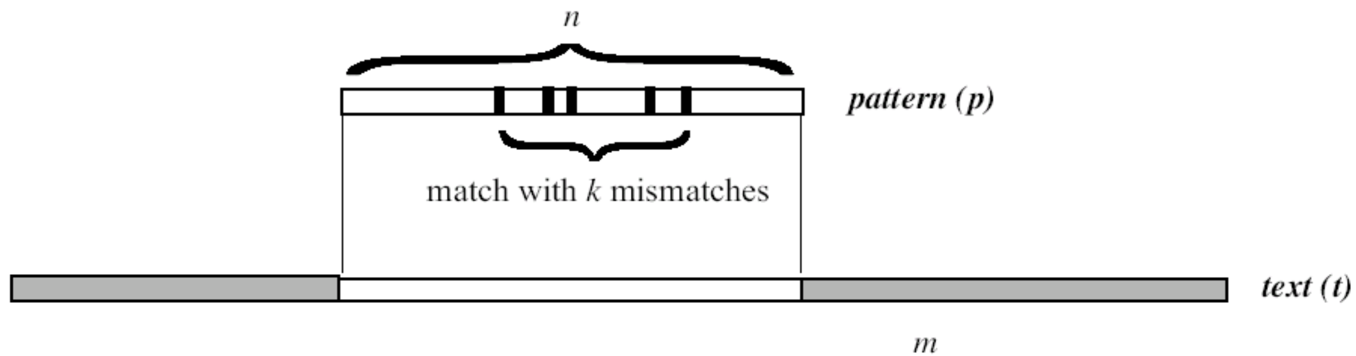
- Goal: Find all substrings of a query that approximately match the text.
- Input: Query $\boldsymbol{q} = q_1 \ldots q_w$

  Text $\boldsymbol{t} = t_1 \ldots t_m$,

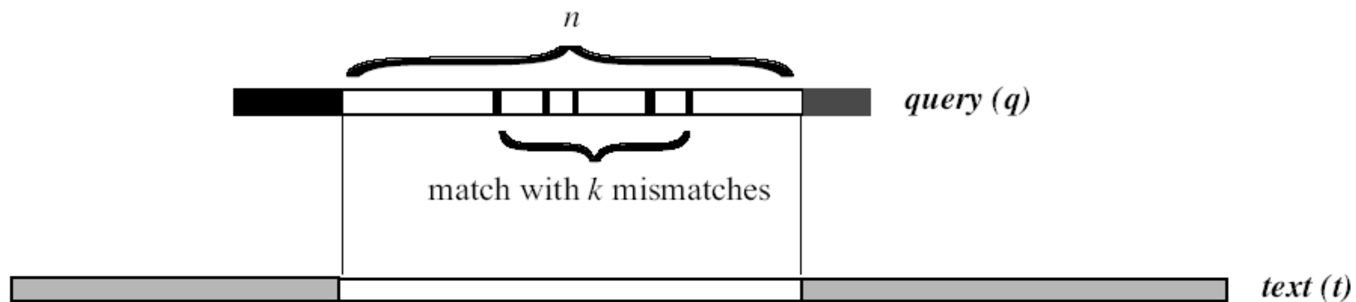
  n = length of matching substrings

  k = maximum number of mismatches
- Output: All pairs of positions (i, j) such that the n-letter substring of query $\boldsymbol{q}$ starting at i approximately matches the n-letter substring of text $\boldsymbol{t}$ starting at j, with at most k mismatches.

Approximate Pattern Matching vs Query Matching



(a) Approximate Pattern Matching



(b) Query Matching

Filtration in Query Matching: Main Idea

- Approximately matching strings share some *perfectly* matching substrings.

- Instead of searching for approximately matching strings (difficult) search for perfectly matching substrings (easy).

MAQ

MAQ is based on spaced seeds; six templates are used, i.e. 11110000, 00001111, 11000011, 00111100, 11001100, and 00110011, where nucleotides at 1 will be indexed while those at 0 are not.

As a threshold between sensibility and efficiency, MAQ does not consider any mapping that has more than two mismatches in the first 28 positions.

MAQ

If a read aligns to multiple positions, MAQ scores the various possibilities through quality scores and compatibility on the complementary strand.

All the hits found on the *forward* strand of the reference sequence are stored in a queue.

While examining the *reverse* strand, if a hit for a read is found, MAQ looks up the queue to check if there is a partial overlapping with one of the hits found on the forward strand.

MAQ

A pair of reads is correctly mapped if both the end of the hits are consistent, i.e. correct orientation within the proper distance.

If only one can be mapped with confidence, a possible scenario is that an indel in one of the two reads occurred.

The classical Smith Waterman algorithm is then applied on the two reads to check validity of the alignment with/without indel or SNP.

MAQ research team is now working on Burrows-Wheeler Aligner (BWA).

Detailed Algorithms

- SHRiMP: Accurate Mapping of Short Color-space Reads [1]
- Bowtie:Ultrafast and memory-efficient alignment of short dna sequences to the human genome [2]

[1] Rumble S, Lacroute P, Dalca AV, Fiume M, Sidow A, Brudno M **SHRiMP: Accurate Mapping of Short Color-space Reads**
PLOS, 2009

[2] Langmead B, Trapnell C, Pop M, and Salzberg SL **Ultrafast and memory-efficient alignment of short dna sequences to the human genome** *Genome Biology*, 2009

SHRiMP

SHRiMP features:

1. Both Color-Space and Letter-Space reads mapping.
2. Allows insertions and deletions.
3. Read mapping probabilities and statistics.

SHRiMP pipeline:

1. Spaced-seed matching
2. Smith-Waterman Algorithm for alignment scores.
3. Alignment probabilities and statistics calculation.

Algoritmi di allineamento di DNA - Smith e Watermann

Smith e Watermann (1981) É una variante dell'algoritmo di N-W che permette di trovare l'allineamento locale ottimo. L'algoritmo é il seguente:

1: **for** (i=0 to length(A)) **do**
2: F(i,0) $\leftarrow$ 0;
3: **for** (j = 0 to length(B)) **do**
4: F(0,j) $\leftarrow$ 0;
5: **for** (i = 1 to length(A)) **do**
6: **for** (j = 0 to length(B)) **do**
7: Choice1 $\leftarrow F(i-1, j-1) + S(A(i), B(j))$;
8: Choice2 $\leftarrow F(i-1, j) - d$;
9: Choice3 $\leftarrow F(i, j-1) - d$;
10: Choice4 $\leftarrow$ 0;
11: F(i,j) $\leftarrow max(Choice1, Choice2, Choice3, Choice4)$;

Smith e Watermann (1981)

- Goal: allineamenti locali, matrici di sostituzione con anche valori negativi;
- Stessi criteri per l'assegnazione degli scores;
- Nuova origine con score zero se i percorsi di origine risulteranno negativi;

L'inizializzazione prevede: $F(i,0) = F(0,j) = 0$ per ogni $i = 1 \ldots n$ e $j = 1 \ldots m$.
In questo caso non ci sono punteggi negativi!
L ' opzione zero corrisponde ad iniziare un nuovo allineamento.
Intuitivamente, se un allineamento ad un certo punto ha un punteggio negativo allora é meglio cominciarne uno nuovo piuttosto che estendere il vecchio.

Una volta costruita la matrice F, l'allineamento si ottiene a partire dall'individuazione del massimo dell'intera matrice e seguendo i puntatori all'indietro fino ad incontrare uno zero, punto d'inizio dell'allineamento. Quindi l'allineamento puó iniziare e terminare in un qualunque punto della matrice.

		T	F	D	E	R	I	L	G	V	Q	T	Y	W	A	E	C	L	A
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Q	0	0	0	2	3	1	0	0	0	0	4	0	0	0	0	3	0	0	0
T	0	3	0	0	2	2	1	0	0	0	0	7	2	0	1	0	1	0	1
F	0	0	12	7	2	0	3	3	0	0	0	2	14	9	4	0	0	3	0
W	0	0	7	5	0	4	0	1	0	0	0	0	9	31	26	21	16	11	6
E	0	0	2	11	9	4	2	0	1	0	3	0	4	26	31	30	25	20	15
C	0	0	0	6	5	5	2	0	0	0	0	1	0	21	26	25	42	37	32
I	0	0	1	1	4	3	10	5	0	4	0	0	0	16	21	24	37	44	39
K	0	0	0	1	1	7	5	7	3	0	5	0	0	11	16	21	32	39	43
G	0	0	0	1	1	2	4	2	12	7	2	5	0	6	12	16	27	34	40
D	0	0	0	4	5	0	0	0	7	10	9	4	1	1	7	16	22	29	35
N	0	0	0	2	5	5	0	0	2	5	11	9	4	0	2	11	17	24	30
A	0	1	0	0	2	3	4	0	1	2	6	12	7	2	2	6	12	19	26
T	0	3	0	0	0	1	3	2	0	1	1	9	9	4	3	2	7	14	21
Y	0	0	10	5	0	0	0	2	0	0	0	4	19	14	9	4	2	9	16

SHRiMP - Smith-Waterman Algorithm

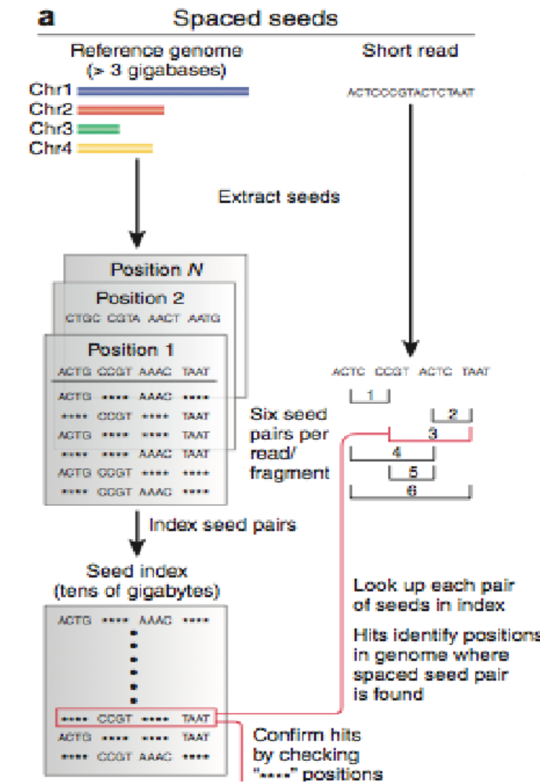
- Sequence 1 = ACACACTA
- Sequence 2 = AGCACACA
- w(match) = +2
- $w(a,-) = w(-,b) = w(mismatch) = -1$

$$H(i,0) = 0,\ 0 \le i \le m$$
$$H(0,j) = 0,\ 0 \le j \le n$$
$$H(i,j) = \max \left\{ \begin{array}{cc} 0 & \\ H(i-1,j-1) + \ w(a_i,b_j) & \text{Match/Mismatch} \\ H(i-1,j) + \ w(a_i,-) & \text{Deletion} \\ H(i,j-1) + \ w(-,b_j) & \text{Insertion} \end{array} \right\},$$

$$H = \begin{pmatrix} & - & A & C & A & C & A & C & T & A \\ - & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ A & 0 & 2 & 1 & 2 & 1 & 2 & 1 & 0 & 2 \\ G & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ C & 0 & 0 & 3 & 2 & 3 & 2 & 3 & 2 & 1 \\ A & 0 & 2 & 2 & 5 & 4 & 5 & 4 & 3 & 4 \\ C & 0 & 1 & 4 & 4 & 7 & 6 & 7 & 6 & 5 \\ A & 0 & 2 & 3 & 6 & 6 & 9 & 8 & 7 & 8 \\ C & 0 & 1 & 4 & 5 & 8 & 8 & 11 & 10 & 9 \\ A & 0 & 2 & 3 & 6 & 7 & 10 & 10 & 10 & 12 \end{pmatrix}$$

SHRiMP pipeline

Spaced seeds:
1111000011110000 , 0000111100001111 1111000000001111,
0000000011111111, 1111111100000000, 0000111111110000



Shrimp uses the seed **111**111** that requires matches at positions 13 and 68, and has length 8 and weight 6. Because seeds with such small weight match extremely often, we require multiple seeds to match within a region before it is further considered, using a technique called **Q-gram filtering**.

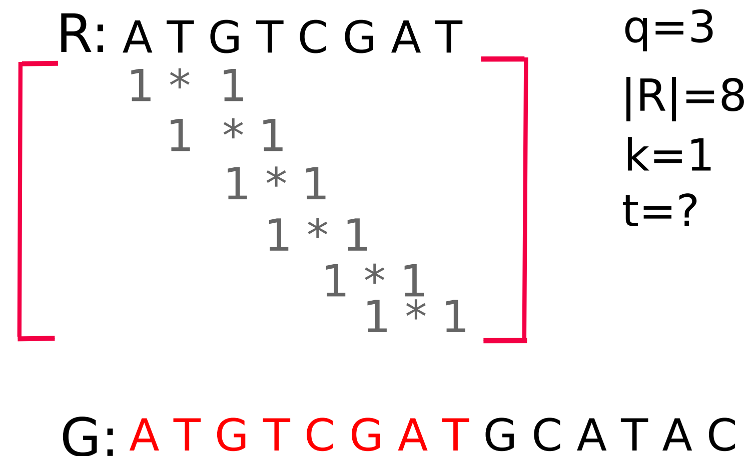
Let us consider a pattern G, a string R, a value k. It is possible to assert that R matches in G with at most k mismatches if it contains at least

$$t = |R| - q + 1 - (kq)$$

where q is the length of substring used to split R.

Gapped q-gram

- gapped q-grams can provide orders of magnitude faster and/or more efficient filtering than contiguous q-grams
- Use substrings with gaps, also called q-shapes
- The optimal threshold could be computed by an exhaustive search of all combinations of k mismatches, but this is prohibitively expensive for large value of k and $|q|$.



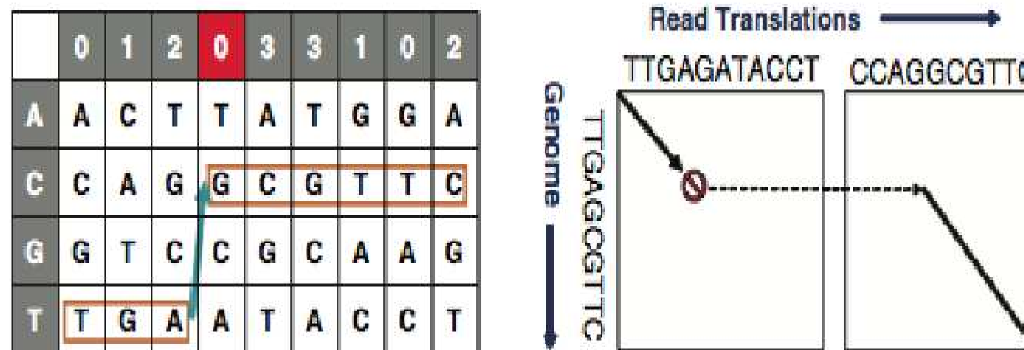
SHRiMP

SHRiMP Pipieline :

- Spaced seed: reads mapping in the genome, look up each read in the genome.
- Q-gram filters: multiple continuos-seeds are used to determine if a good match exists. SHRiMP requires a pre-determined number of seeds from a read to match within a window of the genome before we conduct a thorough comparison.
- Vectorized Smith-Waterman Algorithm.
- Final refinement genome.

SHRiMP modifies the original Smith-Waterman Algorithm by also considering transition from one letter space to another during the search for the optimal alignment (with a penalty for space transition).

$$M_{i,j,k} = \max \begin{cases} M_{i-1,j-1,k} + S(A^k_{i-1}, B_{j-1}) \\ M_{i-1,j,k} - \text{gap} \\ M_{i,j-1,k} - \text{gap} \\ \forall_{n \neq k} M_{i-1,j-1,n} + S(A^k_{i-1}, B_{j-1}) - xover \\ \forall_{n \neq k} M_{i-1,j,n} - \text{gap} - xover \end{cases}$$



SHRiMP - Statistics

SHRiMP estimates the confidence in the possible mappings of each read by using the following statistics:

- **pchance** the probability that the hit occurred by chance
- **pgenome** the probability that the hit was generated by the genome, given the observed rates of the various evolutionary and error events.

For example, a good alignment would have a low pchance (close to 0) and a very high pgenome (close to 1).

SHRiMP - Results

135 million reads of length 35 bp from a single C. savignyi individual.

Highly polymorphic; SNP heterozygosity 4.5%; even small reads can contain several variants.

	SHRiMP	SOLiD Mapper
Uniquely-Mapped Reads	51,856,904 (38.5%)	15,268,771 (11.3%)
Non-Uniquely-Mapped Reads	64,252,692 (47.7%)	12,602,387 (9.4%)
Unmapped Reads	18,657,736 (13.8%)	106,896,174 (79.3%)
Average Coverage (Uniquely-Mapped Reads)	10.3	3.0
Median Coverage (Uniquely-Mapped Reads)	8	1
SNPs	2,119,720	383,099
Deletions (1–5 bp)	51,592	0
Insertions (1–5 bp)	19,970	0