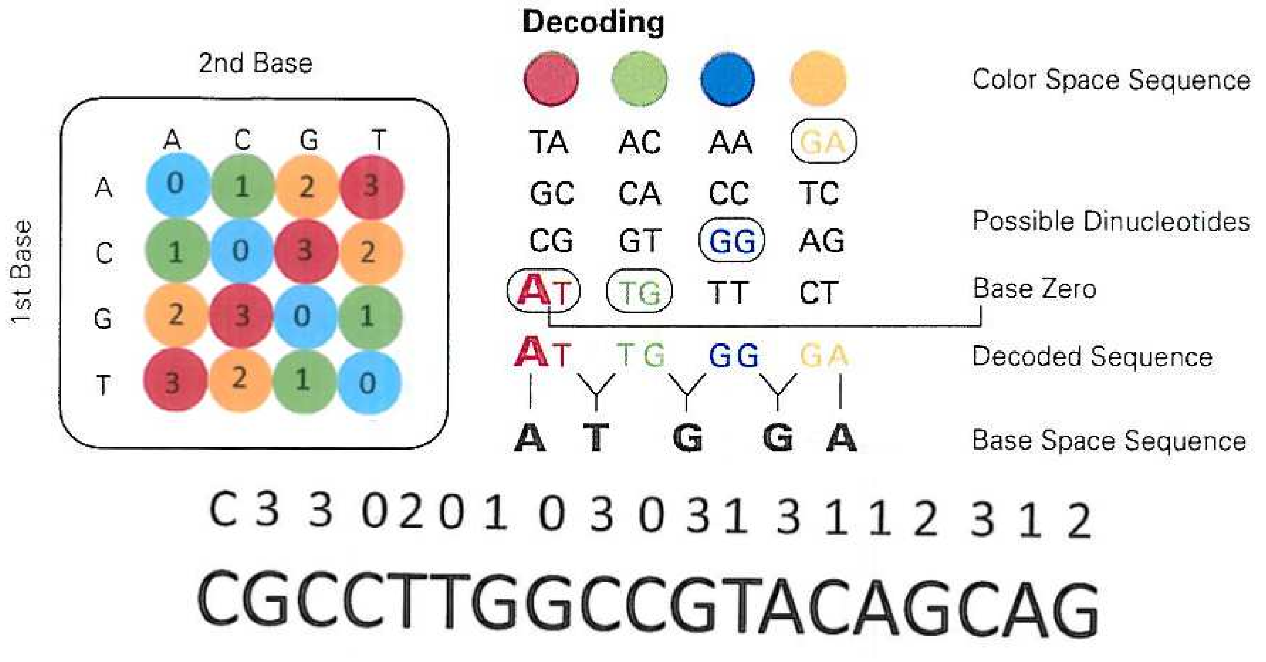
Next generation sequencing

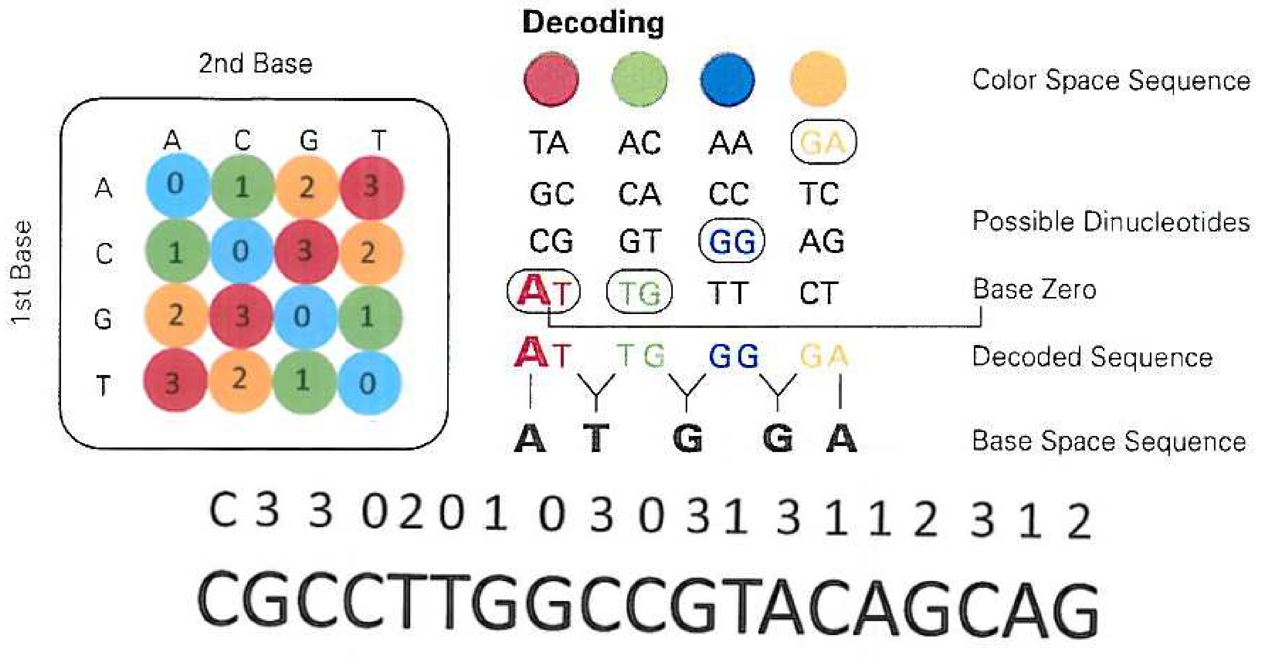
SOLiDfile format

The SOLiDsystem enables massively parallel sequencing of DNA fragments. SOLiDis based on sequential ligation of dye labelled oligonucleotide probes whereby each probe queries **two base position** at a time.



SOLiDfile format

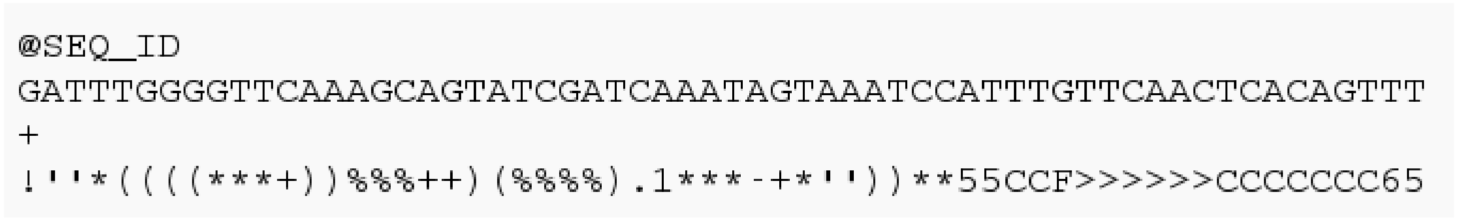
The SOLiDsystem enables massively parallel sequencing of DNA fragments. SOLiDis based on sequential ligation of dye labelled oligonucleotide probes whereby each probe queries **two base position** at a time.



The quality scores are a linear measure of the chance that the color call is correct. In SOLiDtechnology the maximum value of quality is equal to 33 meaning that the sequenced base is correct.

Illumina file format

FASTQ format:



```
@SEQ_ID
GATTTGGGGTTCAAAGCAGTATCGATCAAATAGTAAATCCATTTGTTCAACTCACAGTTT
+
!''*((((***+))%%%++)(%%%%).1***-+*''))**55CCF>>>>>>CCCCCCC65
```

The scores is based on phred scoring scheme:

$$Q_{phred} = -10log_{10}(e)$$

where e is the estimated probability of a base being wrong.

The obtained value is enconded in ASCII character by adding 64 to the Phred value.

Color space: some specifications

String of colors can also be treated as transformations by simply applying one color transformation after another.
To decode the sequence A320:

$f_3(A) \rightarrow$ T

$f_2(T) \rightarrow$ C

continuing in this way until all bases are decoded.

The sequence A32130220 is defined as a single function t:

$$t = f0(f2(f2(f0(f3(f1(f2(f3(A))))))))$$

Color space: some specifications

String of colors can also be treated as transformations by simply applying recursively by the color transformation.
To decode the sequence A320:

$f_3(A) \rightarrow$ T

$f_2(T) \rightarrow$ C

continuing in this way until all bases are decoded.

The sequence A32130220 is defined as a single function t:

$$t = f0(f2(f2(f0(f3(f1(f2(f3(A))))))))$$

Considering the following example:

$$t(A32130) = f0(f3(f1(f2(f3(A))))) \Longrightarrow ATCATT \tag{1}$$

$$t(C32130) = f0(f3(f1(f2(f3(C))))) \Longrightarrow CGACGG \tag{2}$$

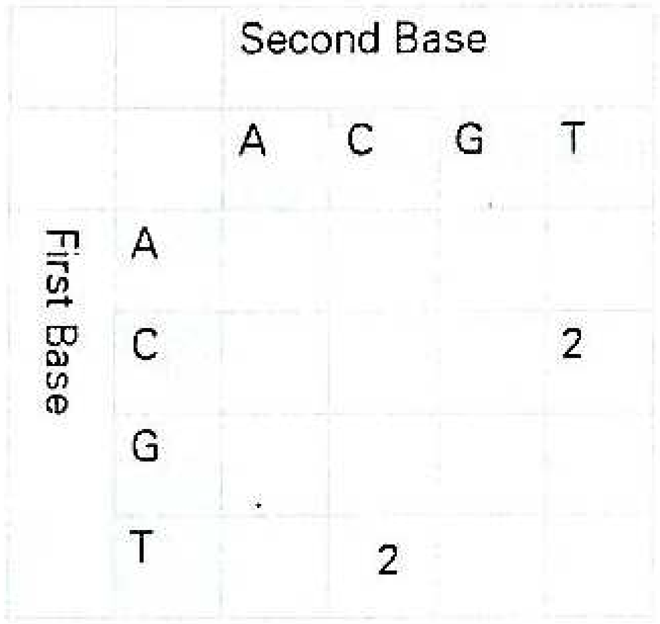
Color space: some specifications

Let $N = \{A, C, G, T\}$ the color-code should satisfy the following requirements:

1. The available colors are 0, 1, 2 and 3:
   Let b, d two nucleotides: color $(bd) \in \{0,\ 1,\ 2,\ 3\}$

Proprierties of color code scheme

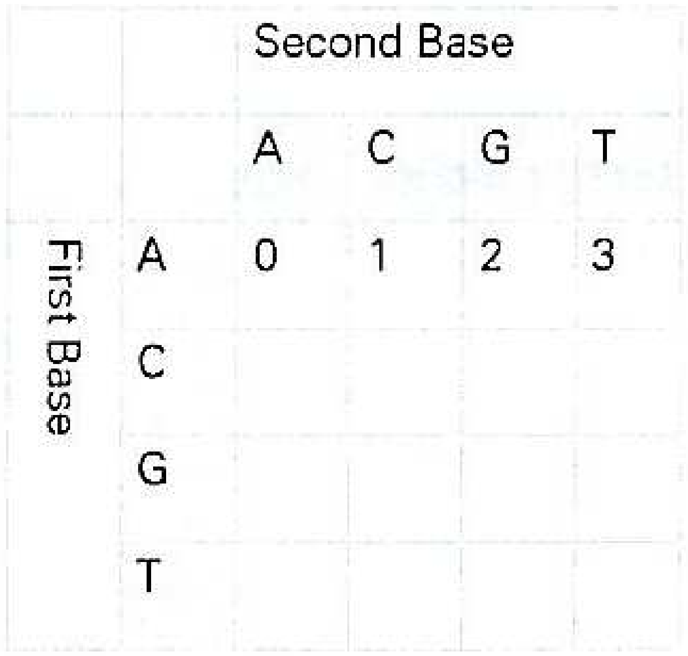
2. A di-base and its reverse get the same color:



		Second Base			
		A	C	G	T
First Base	A				
	C				2
	G				
	T		2		

Proprierties of color code scheme

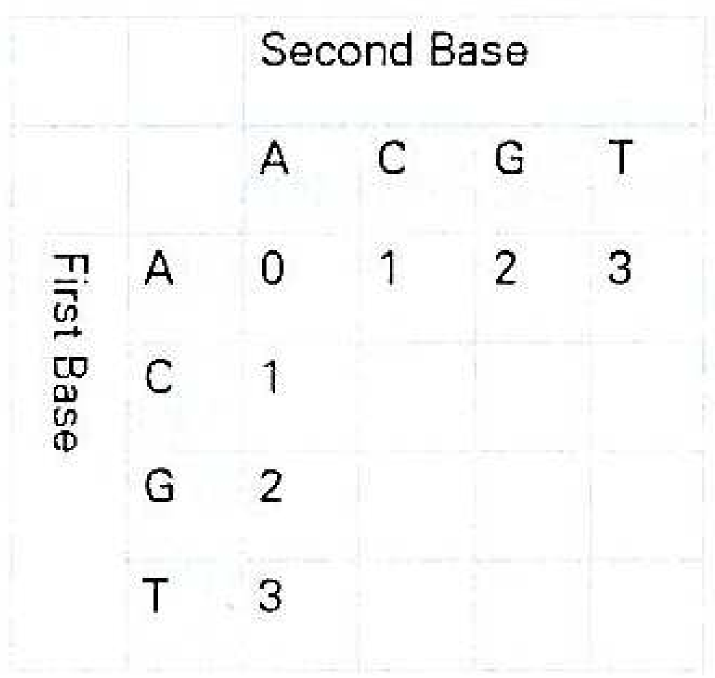
3. Two different di-bases that have the same first base get different colors:



		Second Base			
		A	C	G	T
First Base	A	0	1	2	3
	C				
	G				
	T				

Proprierties of color code scheme

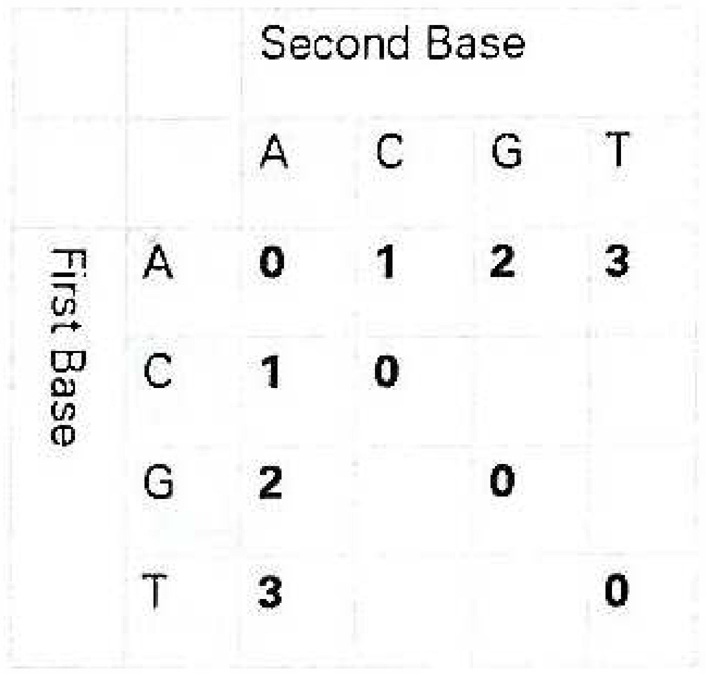
4. Two different di-bases that have the same second base get different colors:



		Second Base			
		A	C	G	T
First Base	A	0	1	2	3
	C	1			
	G	2			
	T	3			

Proprierties of color code scheme

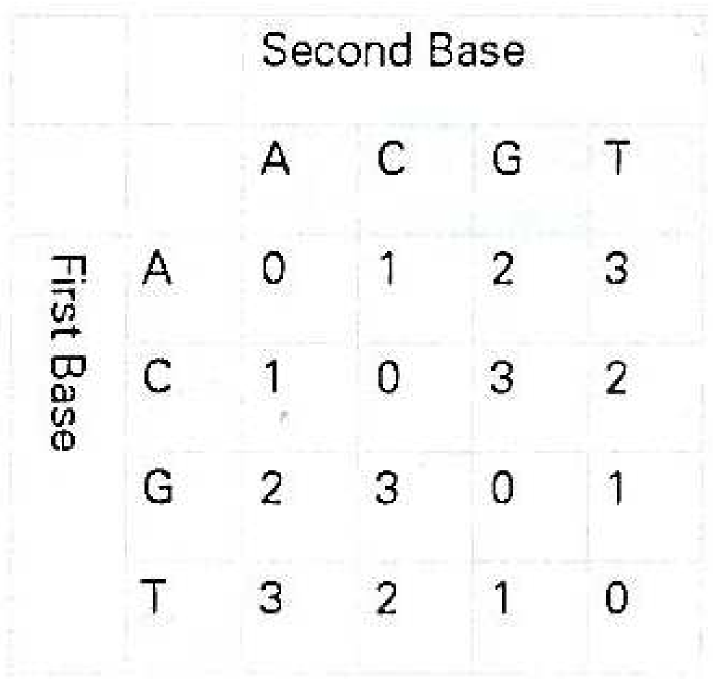
5. Monodibases get the same color:



		Second Base			
		A	C	G	T
First Base	A	0	1	2	3
	C	1	0		
	G	2		0	
	T	3			0

Proprierties of color code scheme

6. A di-base and its complement get the same color:



		Second Base			
		A	C	G	T
First Base	A	0	1	2	3
	C	1	0	3	2
	G	2	3	0	1
	T	3	2	1	0

Color space: some specifications

The group C is isomorphic to the **Klein four-group**.

The elements of Klein group K are :

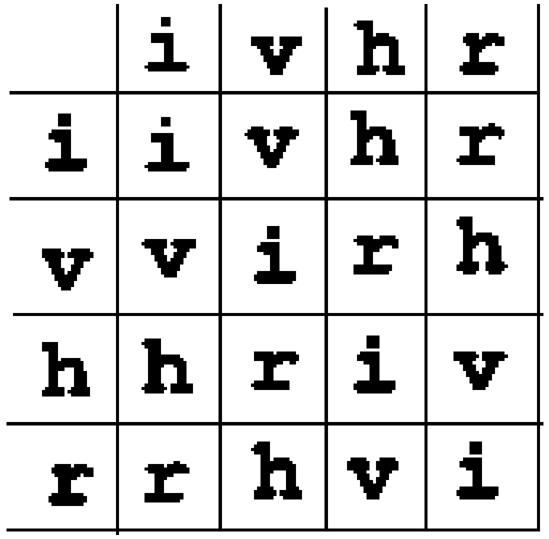
- i identity mapping from the plane to itself
- h reflection in x-axis
- v reflection in y-axis
- r rotation thought 180

and $\circ$ as the binary operation.

The meaningful of the binary operator of a group could be interpreted as "followed by": i.e. $v \circ h = r$ means that a vertical reflection followed by a horizontal reflections is the same, as if it has rotated the rectangle by 180 degrees.

Color space: some specifications

The multiplication table is:

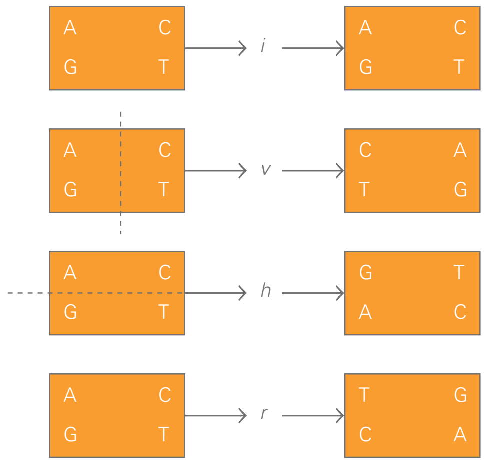


	i	v	h	r
i	i	v	h	r
v	v	i	r	h
h	h	r	i	v
r	r	h	v	i

From the previous example: $v \circ h = r$.

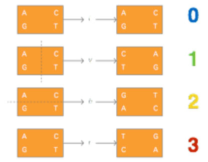
Color space: some specifications

The group of color operations C is isomorphic with respect to Klein four group labelling the corners of the rectangle with the four nucleotides and witnessing their transformations following the four symmetries of a rectangle.



Color space: some specifications

Using the Klein four group to obtain the code for strings of colors as transformation of bases.



Color space: some specifications

The set of color is a Klein four group defines as $C = \{0, 1, 2, 3\}$ where $\oplus$ represents the binary operation.

The four element of the symmetry group of a rectangle can be easily associated with the colors. The corresponding additional table for color sequence is:

$\oplus$	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

Color space: some specifications

The set of color is a Klein four group defines as $C = \{0, 1, 2, 3\}$ where $\oplus$ represents the binary operation.

Since C is a group the following axioms hold:

1. **Closure** C is closed under the operation $\oplus$: $a, b \in C \Longrightarrow a \oplus b \in C$;
2. **Associative** $(a \oplus b) \oplus c = a \oplus (b \oplus c)$ for all $a, b, c \in C$;
3. **Identity** there exists an element $i \in C$ (called the identity of C) such that: $a \oplus i = i \oplus a = a$ for all $a \in C$;
4. **Inverse** for every element $a \in C$, there exists an element $a^{-1} \in C$ (called the inverse of a) such that $a \oplus a^{-1} = a^{-1} \oplus a = i$.

Color space: some specifications

The set of color is a Klein four group defines as $C = 0, 1, 2, 3, 4$ where $\oplus$ represents the binary operation.

Since C is a group the following axioms hold:

1. **Closure** C is closed under the operation $\oplus$: $a, b \in C \Longrightarrow a \oplus b \in C$;
2. **Associative** $(a \oplus b) \oplus c = a \oplus (b \oplus c)$ for all $a, b, c \in C$;
3. **Identity** there exists an element $i \in C$ (called the identity of C) such that: $a \oplus i = i \oplus a = a$ for all $a \in C$;
4. **Inverse** for every element $a \in C$, there exists an element $a^{-1} \in C$ (called the inverse of a) such that $a \oplus a^{-1} = a^{-1} \oplus a = i$.

It is usual, when working with groups, to replace $\oplus$ simply juxtapose the elements being combined.
Then the following sequence A32130220 becames:

$$A \oplus 2 \oplus 1 \oplus 3 \oplus 0 \oplus 2 \oplus 2 \oplus 0$$

Color space: mismatches detection

A *single* polymorphism is amplified in color-space on *two* adjacent color positions.

	Base-space	**Color-space**
		c_1 c_2 c_3 c_4
Reference:	C G **T** A C	C 3 **1** **3** 1
Read:	C G **A** A C	C 3 **2** **0** 1
		c_5 c_6 c_7 c_8

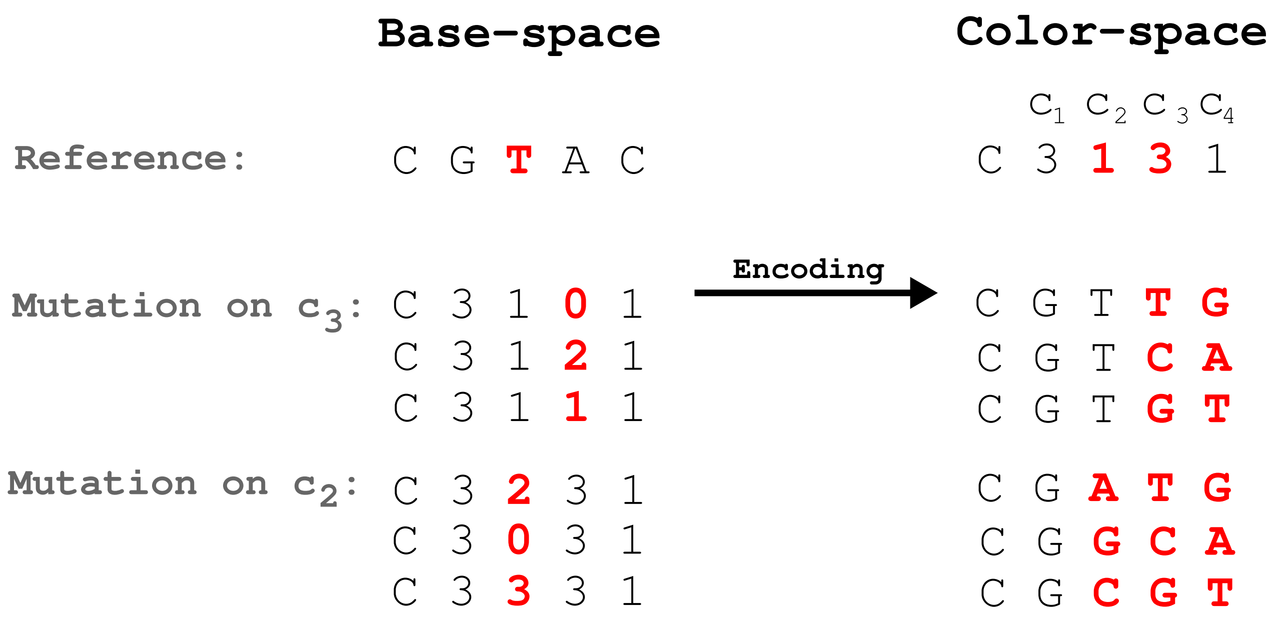
In order for colors c_2, c_3, c_6, c_7 to be consistent with this variant, they must satisfy the following proprieties:

1. $c_2 \neq c_6$
2. $c_2 \oplus c_3 = c_6 \oplus c_7$
3. $c_3 \neq c_7$

Even though there are 9 possible color pairs only three of them correspond to a SNP.

Color space: mismatches detection

If there is only one color muted instead of two, *all* the following sequence from that color is muted.



Color space: mismatches detection

	Base-space	Color-space
		C_1 C_2 C_3 C_4
Reference:	C **G** **T** A C	C **3** **1** **3** 1
Read:	C **A** **A** A C	C **1** **0** **0** 1
		C_5 C_6 C_7 C_8

The three necessary proprieties in order to obtain two adjacent mismatches in the respective nucleotide sequence are:

1. $c_1 \neq c_5$
2. $c_1 \oplus c_2 \neq c_5 \oplus c_6$
3. $c_1 \oplus c_2 \oplus c_3 = c_5 \oplus c_6 \oplus c_7$

Color space: mismatches detection

	Base-space	Color-space
		C_1 C_2 C_3 C_4 C_5
Reference:	C **G T A** C G	C **3 1 3 1** 3
Read:	C **C A T** C G	C **0 1 3 2** 3
		C_6 C_7 C_8 C_9 C_{10}

The proprieties are:

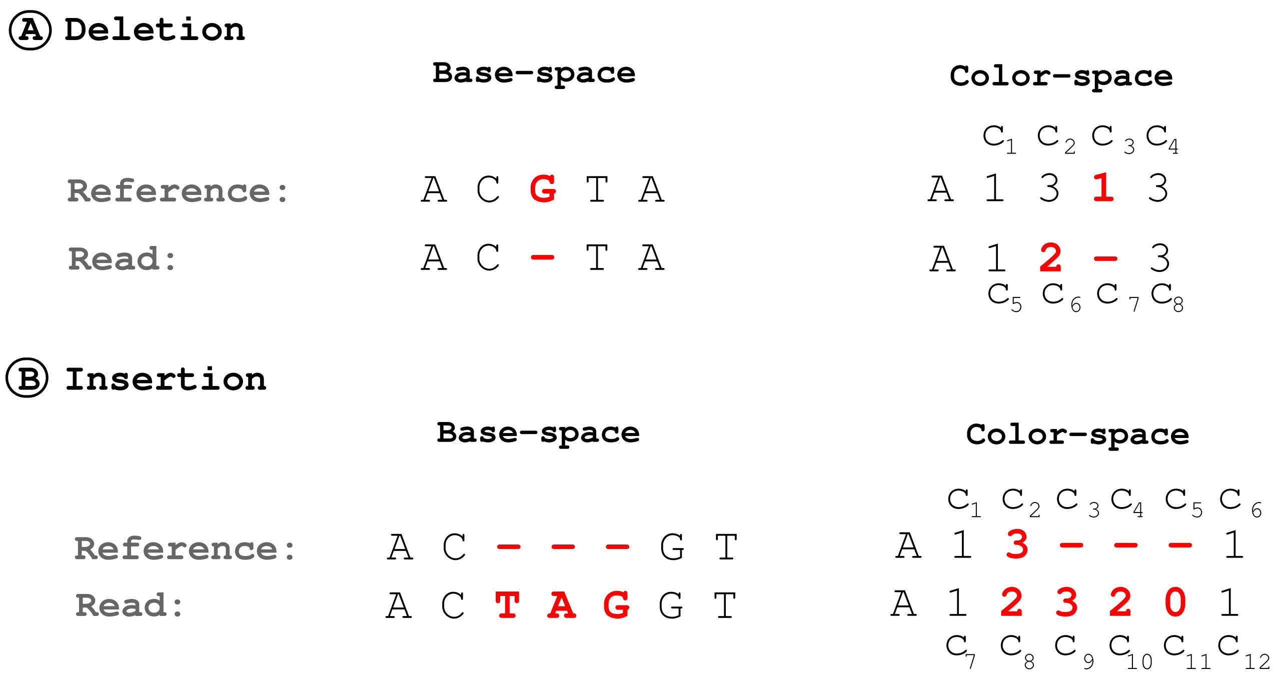
1. $c_1 \neq c_6$
2. $c_1 \oplus c_2 \neq c_6 \oplus c_7$
3. $c_1 \oplus c_2 \oplus c_3 \neq c_6 \oplus c_7 \oplus c_8$
4. $c_1 \oplus c_2 \oplus c_3 \oplus c_4 = c_6 \oplus c_7 \oplus c_8 \oplus c_9$

Color space: formalization

Let $C = \langle b_0, c_1, \ldots, c_n \rangle$ and $C' = \langle b'_0, c'_1, \ldots, c'_n \rangle$ be two color codings where $b_0, b'_0 \in \{ACGT\}$ and $c_i, c'_i \in \{0, 1, 2, 3\}$ then the k-color sub-string $C'_k = \langle c'_j \ldots, c'_{k+j} \rangle$ encodes an isolated (k-1)base change with respect to k-color sub-string $C_k = \langle c_j, \ldots, c_{k+j} \rangle$ iff

- $f_{c_{j-1}} \ldots f_{c_1}(b_0) = f_{c'_{j-1}} \ldots f_{c'_1}(b'_0)$
- $\bigoplus_{i=j}^{j+k-1} c'_i \neq \bigoplus_{i=j}^{j+k-1} c_i$
- $\bigoplus_{i=j}^{j+k} c'_i = \bigoplus_{i=j}^{j+k} c_i$

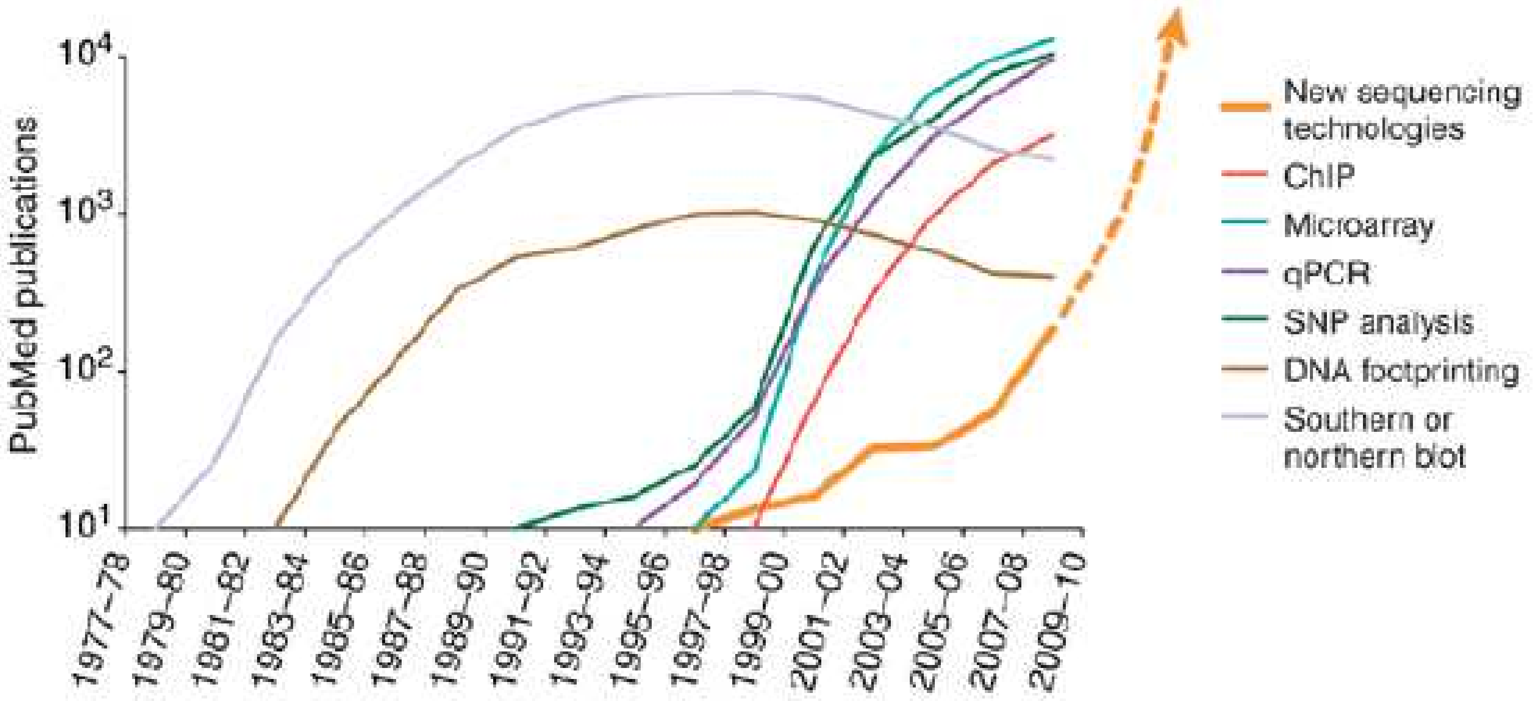
Color space: mismatches detection



In deletion the main propriety is: $c_6 = c_2 \oplus c_3$

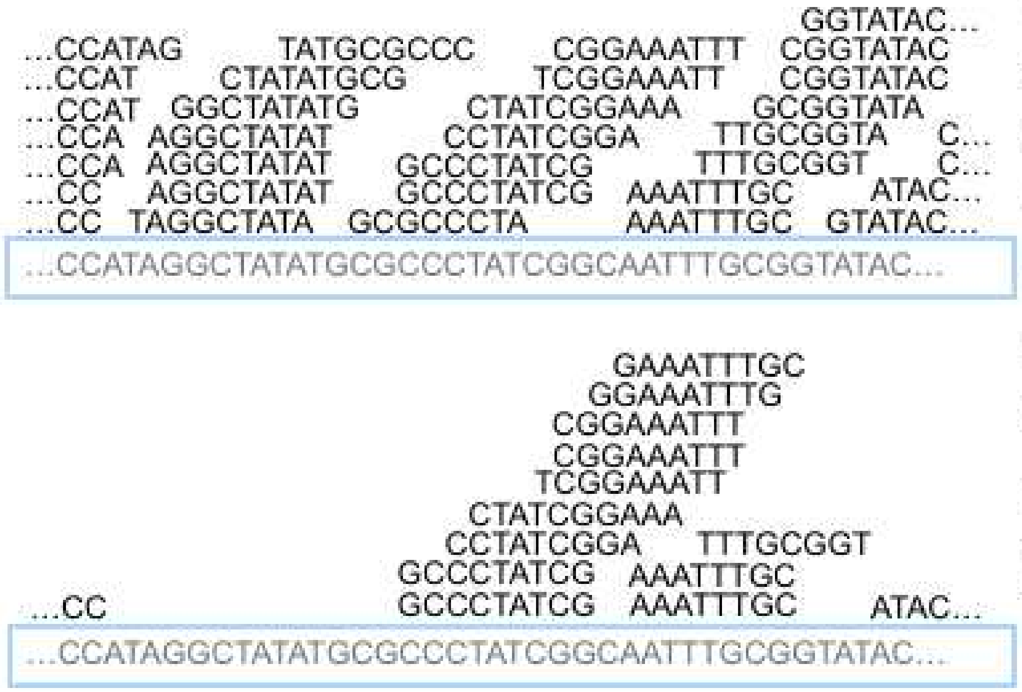
In insertion the main propriety is: $c_2 = c_8 \oplus c_9 \oplus c_{10} \oplus c_{11}$

Next-gen sequencing trends



PubMed was searched in two-year increments for key words and the number of hits plotted over time.

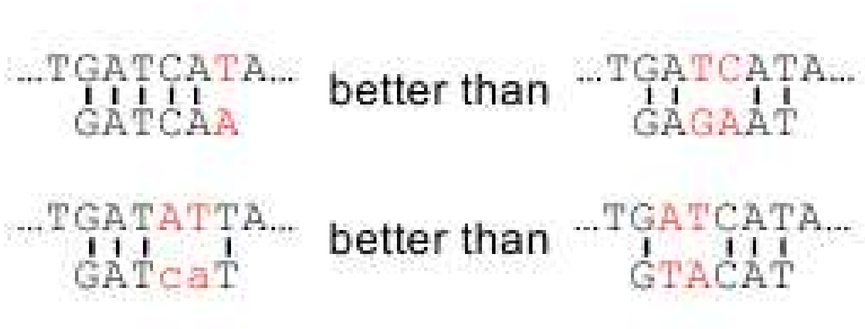
Mapping Short Reads


GGTATAC...
...CCATAG TATGCGCCC CGGAAATTT CGGTATAC
...CCAT CTATATGCG TCGGAAATT CGGTATAC
...CCAT GGCTATATG CTATCGGAAA GCGGTATA
...CCA AGGCTATAT CCTATCGGA TTGCGGTA C...
...CCA AGGCTATAT GCCCTATCG TTTGCGGT C...
...CC AGGCTATAT GCCCTATCG AAATTTGC ATAC...
...CC TAGGCTATA GCGCCCTA AAATTTGC GTATAC...
...CCATAGGCTATATGCGCCCTATCGGCAATTTGCGGTATAC...
GAAATTTGC
GGAAATTTG
CGGAAATTT
CGGAAATTT
TCGGAAATT
CTATCGGAAA
CCTATCGGA TTTGCGGT
GCCCTATCG AAATTTGC
...CC GCCCTATCG AAATTTGC ATAC...
...CCATAGGCTATATGCGCCCTATCGGCAATTTGCGGTATAC...

- Mapping the reads to a reference genome can be computationally challenging
- Hundreds of million of reads, genomes in the size of billion base pairs
- Sequencing errors and genomic variations

Short Reads Alignment

- Where in genome did each read originate? i.e., given a reference genome and a set of reads, report at least one good local alignment for each read if one exists
- What is good? For instance
  - Fewer mismatches is better
  - Failing to align a low-quality base is better than failing to align a high-quality base



String matching

- Short read mapping is a version of a well- known problem: (approximate) string matching
- More than two dozen linear-time algorithms have been designed for exact string matching (KMP, BM, DFA-based, FFT-based,)
- Not as many for approximate strings matching
- But they cannot be used directly for this application

Approximate/Exact string matching

- **Fact** Given two strings w_1 and w_2 at Hamming distance d from each other, they both contain an exact occurrence of a substring of length at least $[m/(d+1)]$. [Baeza-Yates and Perleberg, or BYP]

- **Example**
  w_1 = **GATTTCA**
  w_2 = **GGTTACA**
  TT and **CA** are occurring exactly.
  In fact $[m/(d+1)] = [7/3] = 2$.

Mapping algorithms

All alignment tools considered follow a similar organization into macro s.eps:

- Pre-processing to pre-process the reference genome (or reads set) into one or more index tables;
- Mapping to find matches in the index tables for all queried subsequences and to locate the regions of potential homology;
- Result Refinement to examine all potential matches produced in the previous step using the full read-genome substring alignment.

Available algorithms List

- MAQ `http://maq.sourceforge.net/` (PMID: 18714091).

- ELAND
- ZOOM `http://www.bioinformaticssolutions.com/products/zoom/index.php` (PMID: 18684737)
- MOSAIK `http://bioinformatics.bc.edu/marthlab/Mosaik`.

- SOCS `http://socs.biology.gatech.edu/` (PMID: 18842598)
- PERM `http://code.google.com/p/perm/` (PMID: 19675096)
- SHRiMP `http://compbio.cs.toronto.edu/shrimp/` (PMID: 19461883).

- Bowtie `http://bowtie-bio.sourceforge.net`
- BWA `http://bio-bwa.sourceforge.net/bwa.shtml` (PMID: 19451168)

Section 1: Repeat Finding and Hash Tables

Genomic Repeats

- **Repeat**: A sequence of DNA that occurs more than once in a genome.

- **Example**: ATGGTCTAGGTCCTAGTGGTC

Genomic Repeats

- **Repeat**: A sequence of DNA that occurs more than once in a genome.
- **Example**: ATGGTCTAGGTCCTAGTGGTC

Genomic Repeats

- **Repeat**: A sequence of DNA that occurs more than once in a genome.
- **Example**: ATGGTCTAGGTCCTAGTGGTC

Genomic Repeats

- **Repeat**: A sequence of DNA that occurs more than once in a genome.

- **Example**: ATGGTCTAGGTCCTAGTGGTC

Genomic Repeats

- **Repeat**: A sequence of DNA that occurs more than once in a genome.

- **Example**: ATGGTCTAGGTCCTAGTGGTC

- Why do we want to find repeats?
  1. Understand more about evolution.
  2. Many tumors are characterized by an explosion of repeats.
  3. Genomic rearrangements are often associated with repeats.

Genomic Repeats

- **Note**: Often, a repeat will refer to a segment of DNA that occurs very often *with slight modifications*.
- As we might imagine, this is a more difficult computational problem.

Genomic Repeats

- **Note**: Often, a repeat will refer to a segment of DNA that occurs very often *with slight modifications*.

- As we might imagine, this is a more difficult computational problem.

- **Example**: Say our target segment is GGTC and we allow one substitution:
    - ATGGTCTAGGACCTAGTGTTC

Genomic Repeats

- **Note**: Often, a repeat will refer to a segment of DNA that occurs very often *with slight modifications*.
- As we might imagine, this is a more difficult computational problem.
- **Example**: Say our target segment is GGTC and we allow one substitution:
  - ATGGTCTAGGACCTAGTGTTC

Genomic Repeats

- **Note**: Often, a repeat will refer to a segment of DNA that occurs very often *with slight modifications*.

- As we might imagine, this is a more difficult computational problem.

- **Example**: Say our target segment is GGTC and we allow one substitution:
  - ATGGTCTAGGACCTAGTGTTC

Genomic Repeats

- **Note**: Often, a repeat will refer to a segment of DNA that occurs very often *with slight modifications*.
- As we might imagine, this is a more difficult computational problem.
- **Example**: Say our target segment is GGTC and we allow one substitution:
  - ATGGTCTAGGACCTAGTGTTC

Extending *l*-mer Repeats

- Long repeats are difficult to find, but short repeats are easy.

- Simple approach to finding long repeats:
  1. Find exact repeats of short *l*-mers (*l* is usually 10 to 13).
  2. Use *l*-mer repeats to potentially extend into longer, **maximal** repeats.

Extending *l*-mer Repeats

- **Example**: We start with a repeat of length 4.
- GCTTACAGATTCAGTCTTACAGATGGT

Extending *l*-mer Repeats

- **Example**: We start with a repeat of length 4.

- GCTTACAGATTCAGTCTTACAGATGGT

Extending *l*-mer Repeats

- **Example**: We start with a repeat of length 4.
- GCTTACAGATTCAGTCTTACAGATGGT
- Extend both these 4-mers:
- GCTTACAGATTCAGTCTTACAGATGGT

Extending *l*-mer Repeats

- **Example**: We start with a repeat of length 4.
- GCTTACAGATTCAGTCTTACAGATGGT
- Extend both these 4-mers:
- GCTTACAGATTCAGTCTTACAGATGGT

Extending *l*-mer Repeats

- **Example**: We start with a repeat of length 4.
- GCTTACAGATTCAGTCTTACAGATGGT
- Extend both these 4-mers:
- GCTTACAGATTCAGTCTTACAGATGGT

Extending *l*-mer Repeats

- **Example**: We start with a repeat of length 4.
- GCTTACAGATTCAGTCTTACAGATGGT
- Extend both these 4-mers:
- GCTTACAGATTCAGTCTTACAGATGGT

Extending *l*-mer Repeats

- **Example**: We start with a repeat of length 4.
- GCTTACAGATTCAGTCTTACAGATGGT
- Extend both these 4-mers:
- GCTTACAGATTCAGTCTTACAGATGGT

Extending *l*-mer Repeats

- **Example**: We start with a repeat of length 4.
- GCTTACAGATTCAGTCTTACAGATGGT
- Extend both these 4-mers:
- GCTTACAGATTCAGTCTTACAGATGGT

Extending *l*-mer Repeats

- **Example**: We start with a repeat of length 4.
- GCTTACAGATTCAGTCTTACAGATGGT
- Extend both these 4-mers:
- GCTTACAGATTCAGTCTTACAGATGGT

Extending *l*-mer Repeats

- **Example**: We start with a repeat of length 4.
- GCTTACAGATTCAGTCTTACAGATGGT
- Extend both these 4-mers:
- GCTTACAGATTCAGTCTTACAGATGGT
- Maximal repeat: CTTACAGAT

Maximal Repeats: Issue

- In order to find maximal repeats in this way, we need ALL start locations of ALL *l*-mers in the genome.

- **Hashing** lets us find repeats quickly in this manner.

Hashing

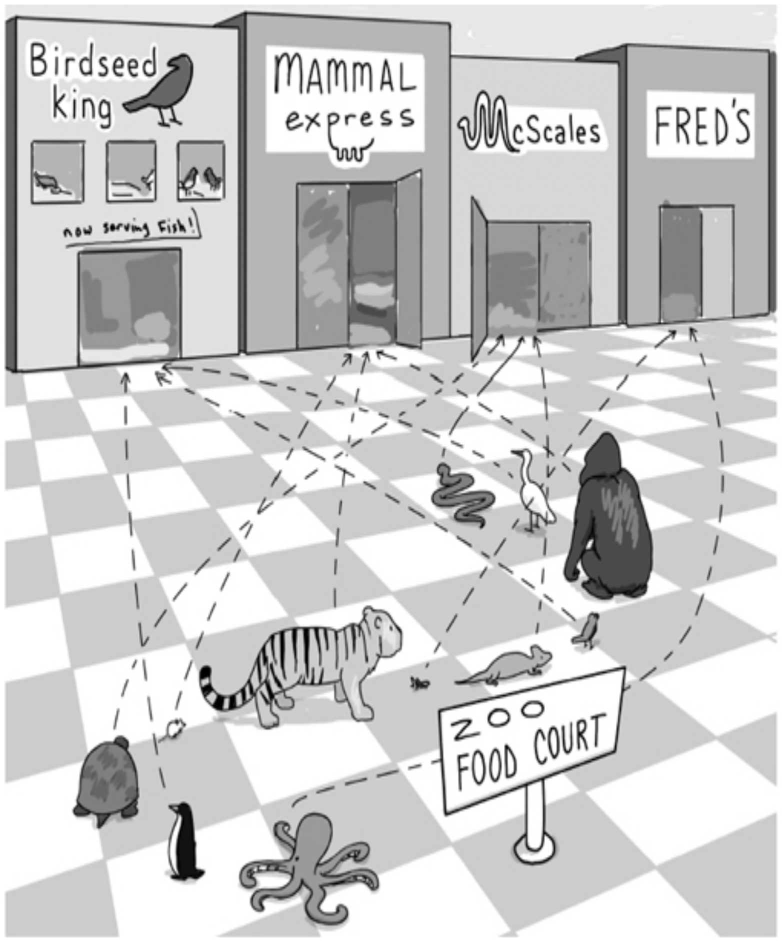
- Hashing is a very efficient way to store and retrieve data.
- What does hashing do?
- Say we are given a collection of data.
- For different entry, generate a unique integer and store the entry in an array at this integer location.

Hashing: Definitions

- **Records**: data stored in a *hash table*.
- **Keys**: Integers identifying set of *records*.
- **Hash Table**: The array of keys used in hashing.
- **Hash Function**: Assigns each record to a key.
- **Collision**: Occurs when more than one record is mapped to the same index in the hash table.

Hashing: Example

- Records: Animals
- Keys: Where each animal eats.



Records x	Keys $h(x)$
Penguin	1
Octopus	4
Turtle	3
Mouse	2
Snake	3
Heron	1
Tiger	2
Iguana	3
Ape	2
Cricket	4
Sparrow	1

Hashing DNA sequences

- Each *l*-mer can be translated into a binary string:
  - **A** can be represented as **00**
  - **T** can be represented as **01**
  - **G** can be represented as **10**
  - **C** can be represented as **11**
  - **Example**: ATGCTA = 000110110100

- After assigning a unique integer to each *l*-mer, it is easy to obtain all start locations of each *l*-mer in a genome.

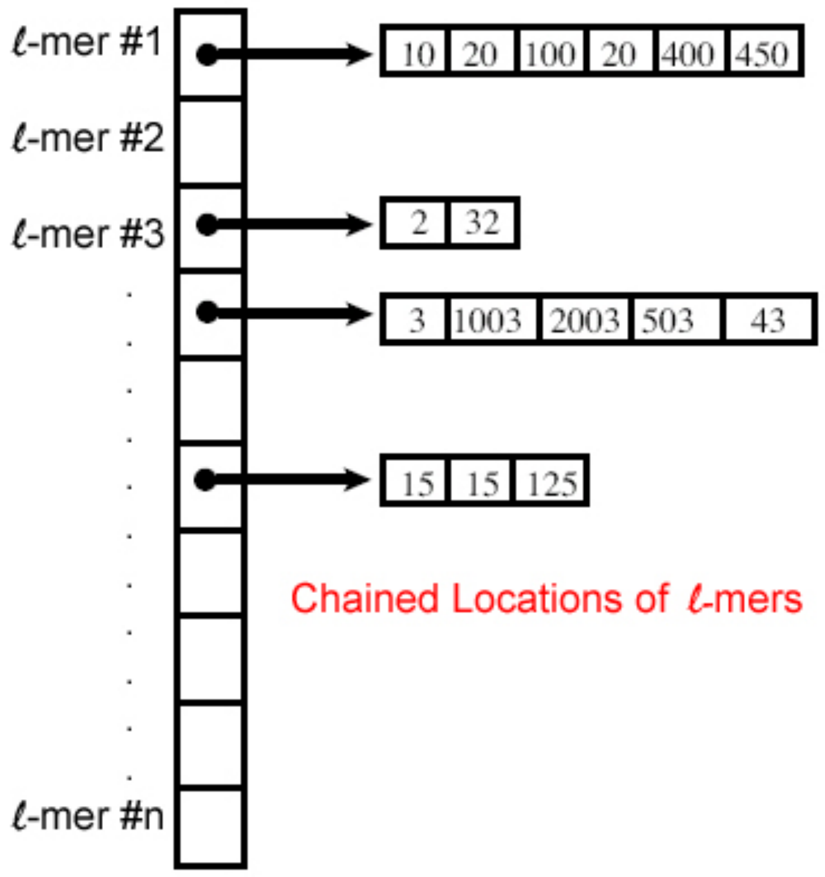
Hashing: Maximal Repeats

- To find repeats in a genome:
  1. For all *l*-mers in the genome, note the start position and the sequence.
  2. Generate a hash table index for each unique *l*-mer sequence.
  3. At each index of the hash table, store all genome start locations of the *l*-mer that generated the index.
  4. Anywhere there are collisions in the table, extend corresponding *l*-mers to maximal repeats.

- How do we deal with collisions?

Hashing: Collisions

- In order to deal with collisions:
- "Chain" all start locations of *l*-mers.
- This can be done via a data structure called a **linked list**.



Hash Table

An *hash table* is basically an array which uses a hash function to efficiently map an input string, called *key* to an associated value.

Each input string, typically large, is encoded by a hash function in a smaller datum, called hash, (i.e. a single integer).

A hash function is a function which converts every string into a numeric value, called its hash value; for example, we might have:

$$hash(''hello'') = 5$$

Hash Table

First application way

The hash function applied to a read r will return the index of the array that contains the *positions* in the reference in which the read is found.

If the hash table stores the position of each substring of 35 nucleotides of a reference genome, and if the same substring appears more than once in the reference, then the hash table is **not** an array of integers, but an array of sets of integers.

Ideally, the hash function should map each possible key to a different index; but this is not ensured in practice: two or more different keys may have the same hash **hash collision**.

Hash Table

In the genome mapping step if the hash table contains positions that match different substrings, a further step is required to ensure the *mapping correctness*.

The hash table can contains the information of reference genome or of the input reads, so that a full mapping can be performed by the lookup of a read (or a set of reads) in the hash table of the reference genome, or vice versa.

In both cases, due to hash collisions, **all the found matches have to be verified**.

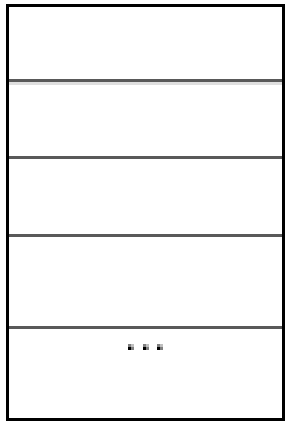
Hash Table

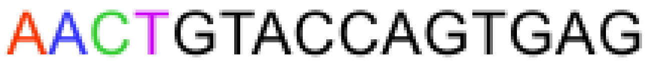
Second application way

- k-mers are substrings of length k (also called seeds or q-grams)
- A direct-address table of all k-mers over a DNA alphabet requires 4^k entries (some could be unused)
- Each entry in the table is a pointer to the list of occurrences of that k-mer
- Hash the genome or the queries (short reads)?

Hashing k-mers

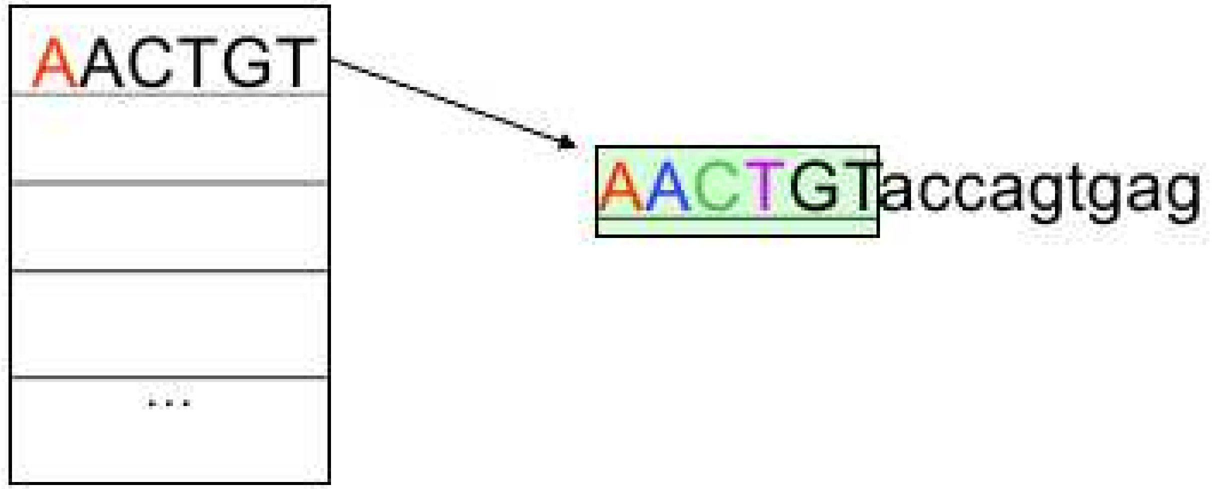
Create a hash table of size 4^k





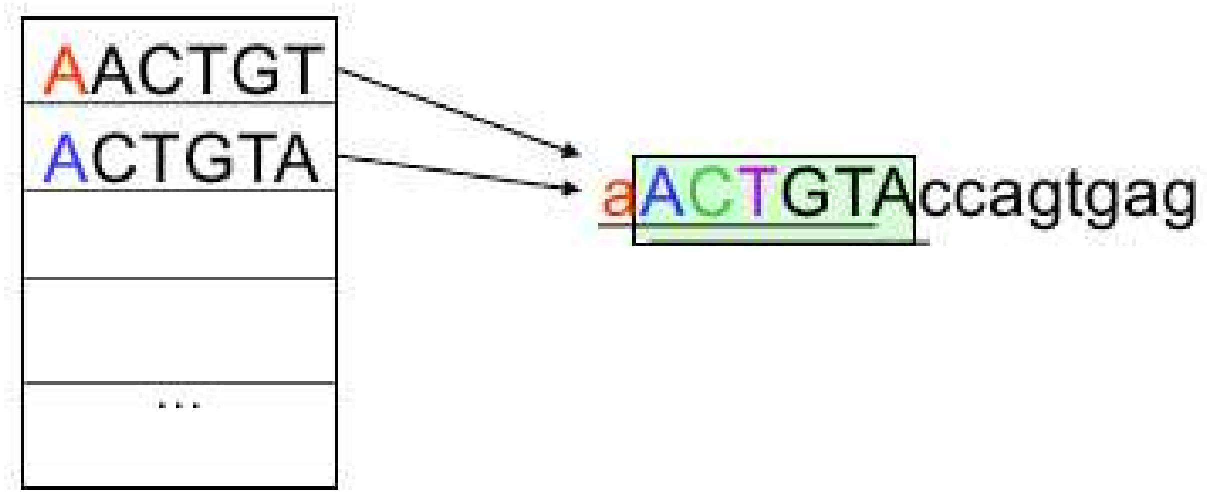
Hashing k-mers

Create a hash table of size 4^k



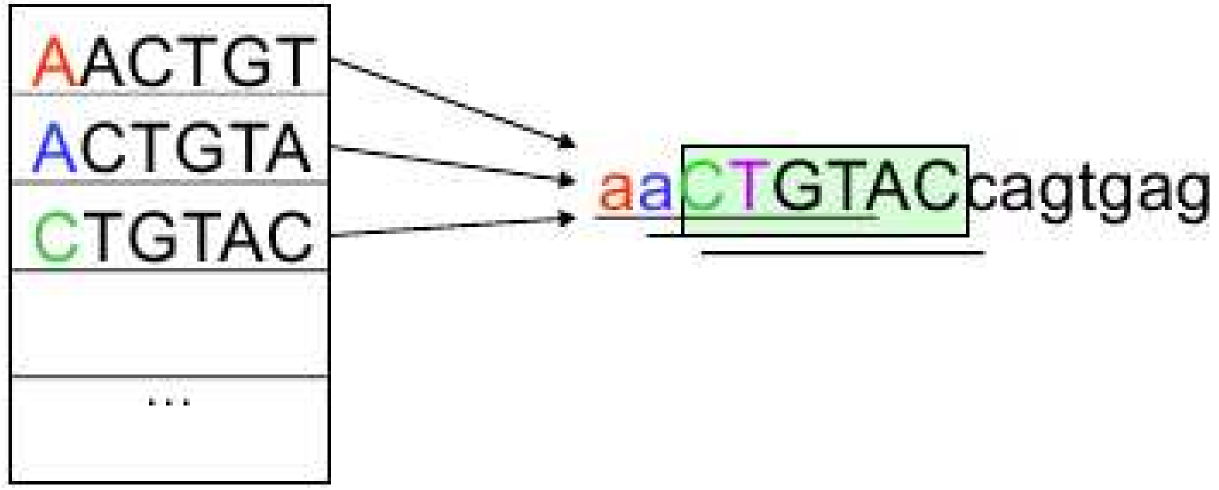
Hashing k-mers

Create a hash table of size 4^k



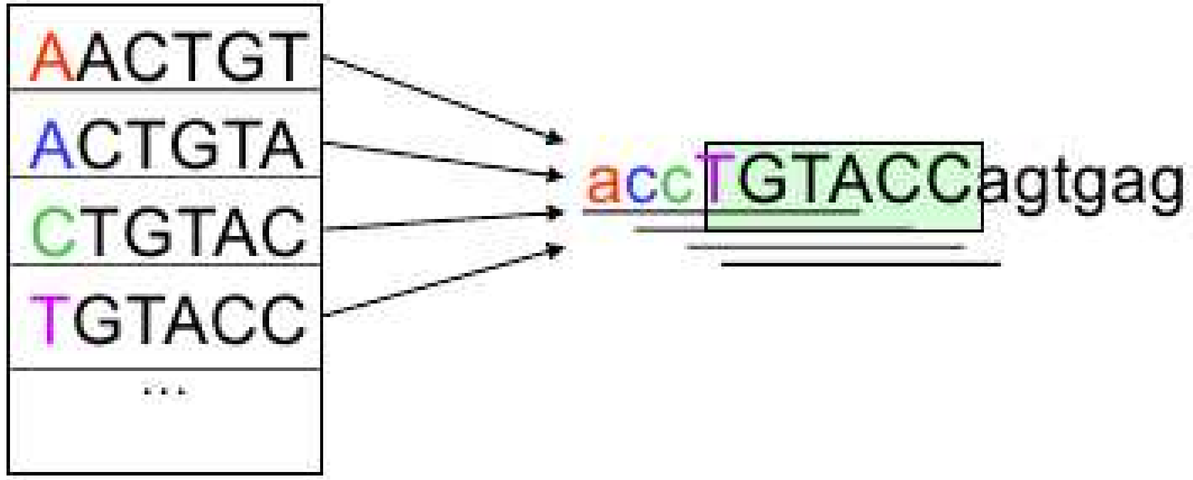
Hashing k-mers

Create a hash table of size 4^k



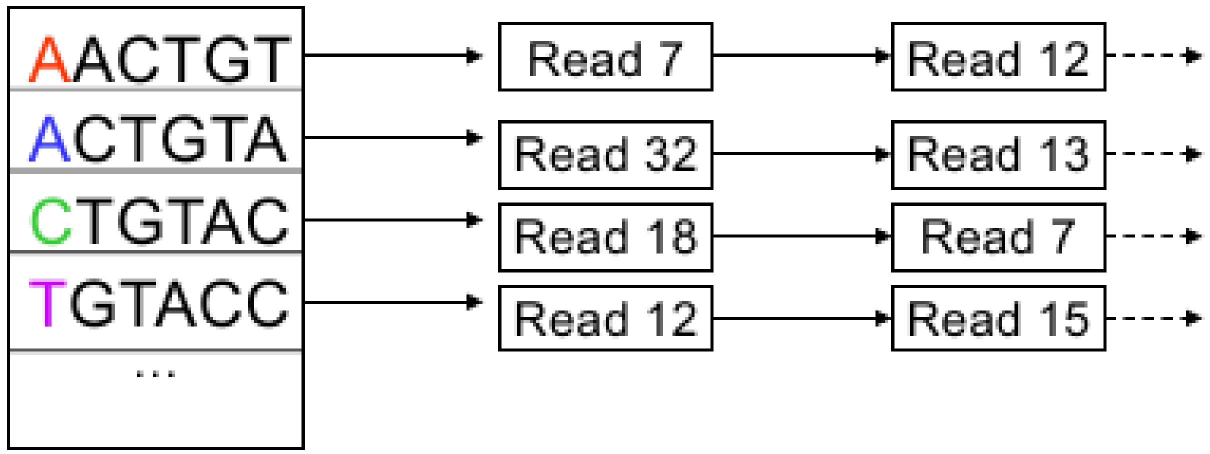
Hashing k-mers

Create a hash table of size 4^k



Hashing k-mers

Create a hash table of size 4^k: each entry has a pointer to the list of reads that contain that k-mer



Hashing k-mers

Considerations

- The table can be very large
- The number of entries should be as uniform as possible (avoid sparseness)
- Entries in direct-address tables tend to be non- uniform, but if one uses hashing functions (e.g., $h(x) =$ x mod p, p prime) collisions need to be resolved

BLAST (Altschul et al, J Mol Biol 1990)

BLAST is faster than Smith-Waterman, but it cannot guarantee the optimal alignments of the query and database sequences. BLAST essentially follows the seed-and-extend paradigm.

- keep the position of each n-mer subsequence
- scan the database sequences for n-mer exact matches, **seeds**
- extend and joins the seeds first without gaps
- refine them by a $SmithWaterman$ algorithm

Seeds idea

The idea of seeds is used for preprocessing and matching.

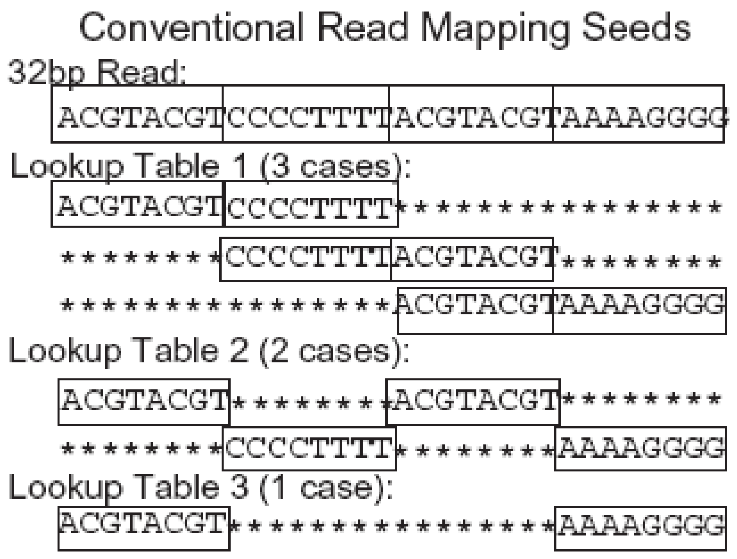
A **seed** is a set of selected positions within a window which generates fixed length sub-sequences along a string.

When a seed is aligned to a read or a genomic sequence, selected positions are concatenated to form a fixed length subsequence which can be used to extract similar reads or genomic substrings.

Each seed is associated with a weight that correspond to the probability of seed matching by chance or not, when the seed is very short is associated with a *low weight* since the probability that matching by chance becomes greater.

Seeds idea

All the algorithms that use the seeds divide each read into $k + m$ fragments to provide full sensitivity to k mismatches.

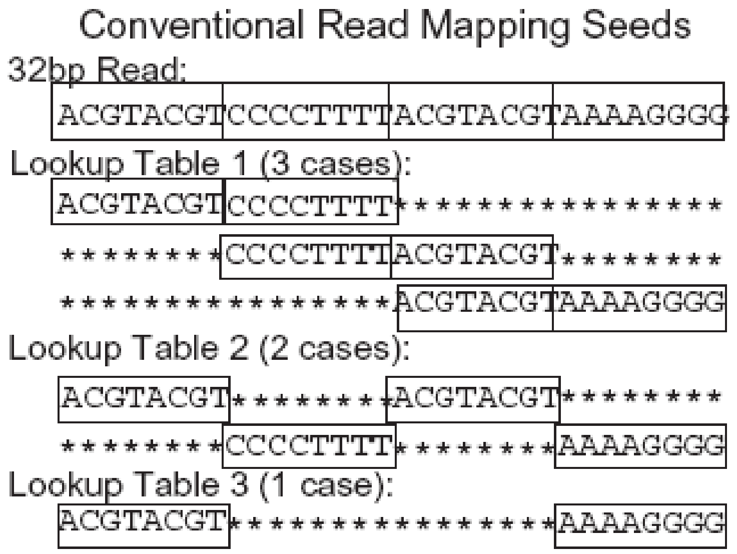


Conventional Read Mapping Seeds
32bp Read:
ACGTACGT CCCCTTTT ACGTACGT AAAAGGGG
Lookup Table 1 (3 cases):
ACGTACGT CCCCTTTT ****************
******** CCCCTTTT ACGTACGT ********
**************** ACGTACGT AAAAGGGG
Lookup Table 2 (2 cases):
ACGTACGT ******** ACGTACGT ********
******** CCCCTTTT ******** AAAAGGGG
Lookup Table 3 (1 case):
ACGTACGT **************** AAAAGGGG

Conventional seeds divide a 32 bp read into four substrings. For any alignment within two mismatches, at least one of six pairs of substrings will match exactly.

Seeds idea

All the algorithms that use the seeds divide each read into $k + m$ fragments to provide full sensitivity to k mismatches.


Conventional Read Mapping Seeds
32bp Read:
ACGTACGT CCCCTTTT ACGTACGT AAAAGGGG
Lookup Table 1 (3 cases):
ACGTACGT CCCCTTTT ****************
******** CCCCTTTT ACGTACGT ********
**************** ACGTACGT AAAAGGGG
Lookup Table 2 (2 cases):
ACGTACGT ******** ACGTACGT ********
******** CCCCTTTT ******** AAAAGGGG
Lookup Table 3 (1 case):
ACGTACGT **************** AAAAGGGG

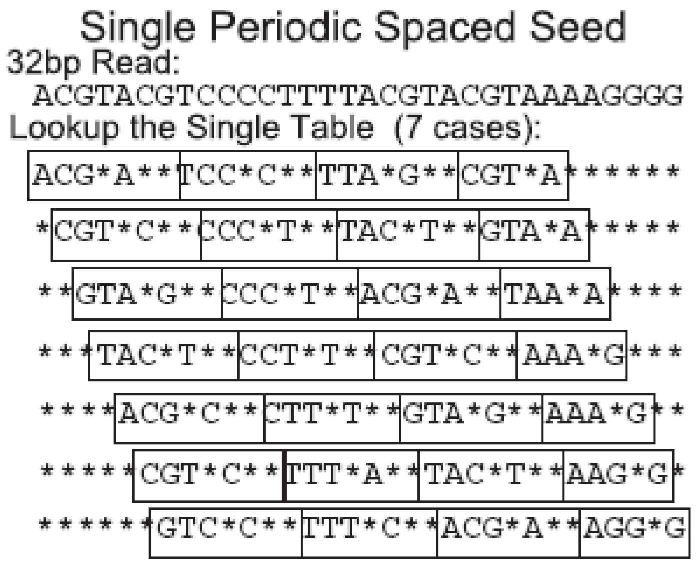
The choice of different algorithms:

- *Corona Lite* chooses m = 3;
- *ELAND, MAQ and SOAP* chooses m = 2.

Spaced Seeds idea

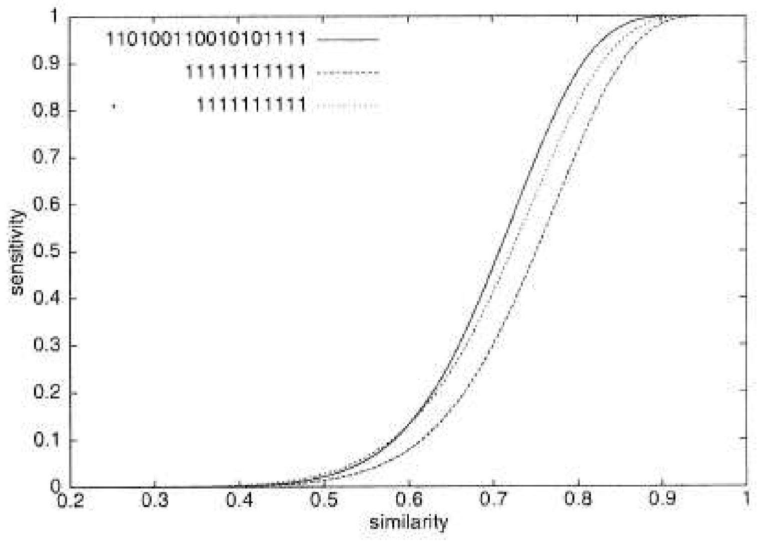
- More recently *space* seeds, where *predetermined* positions in the read are allowed not to match, have been shown to be more sensitive.
- Space seed is a set of 'care' and 'don't care' positions, annotated as '1's and '*'s respectively. Usually the length of the seed is the total length of the string, and the weight of the seed as the number of 1s in the string.
- It was empirically demonstrated that an optimal spaced seed quadruples the search speed, without sacrificing sensitivity (probability of having at least one hit in a highly similar region between query and genome) Several results followed to explain the advantage

PerM - spaced seeds definition


Single Periodic Spaced Seed
32bp Read:
ACGTACGTCCCCTTTTACGTACGTAAAAGGGG
Lookup the Single Table (7 cases):
ACG*A**TCC*C**TTA*G**CGT*A******
*CGT*C**CCC*T**TAC*T**GTA*A*****
GTA*GCCC*T**ACG*A**TAA*A****
TAC*T**CCT*T**CGT*C**AAA*G
****ACG*C**CTT*T**GTA*G**AAA*G**
*****CGT*C**TTT*A**TAC*T**AAG*G*
******GTC*C**TTT*C**ACG*A**AGG*G

The single periodic spaced seed full sensitive to two mismatches over a 32 bp read. For any alignment within two mismatches, at least one out of the seven subsequences will match exactly. This seed is composed of repeating the pattern (111*1**)

Spaced Seeds idea



Performance of weight 11 spaced seeds versus weight 11 and 10 consecutive seeds (from Ma et al, Bioinformatics 2002)