Mapping on reference genomes

Stefan Simm
simm@bio.uni-frankfurt.de
2019

Literature

Mapping short DNA sequencing reads and calling variants using mapping quality scores. Li et al. Genome Research (2008)
doi: 10.1101/gr.078212.108 (MAQ)

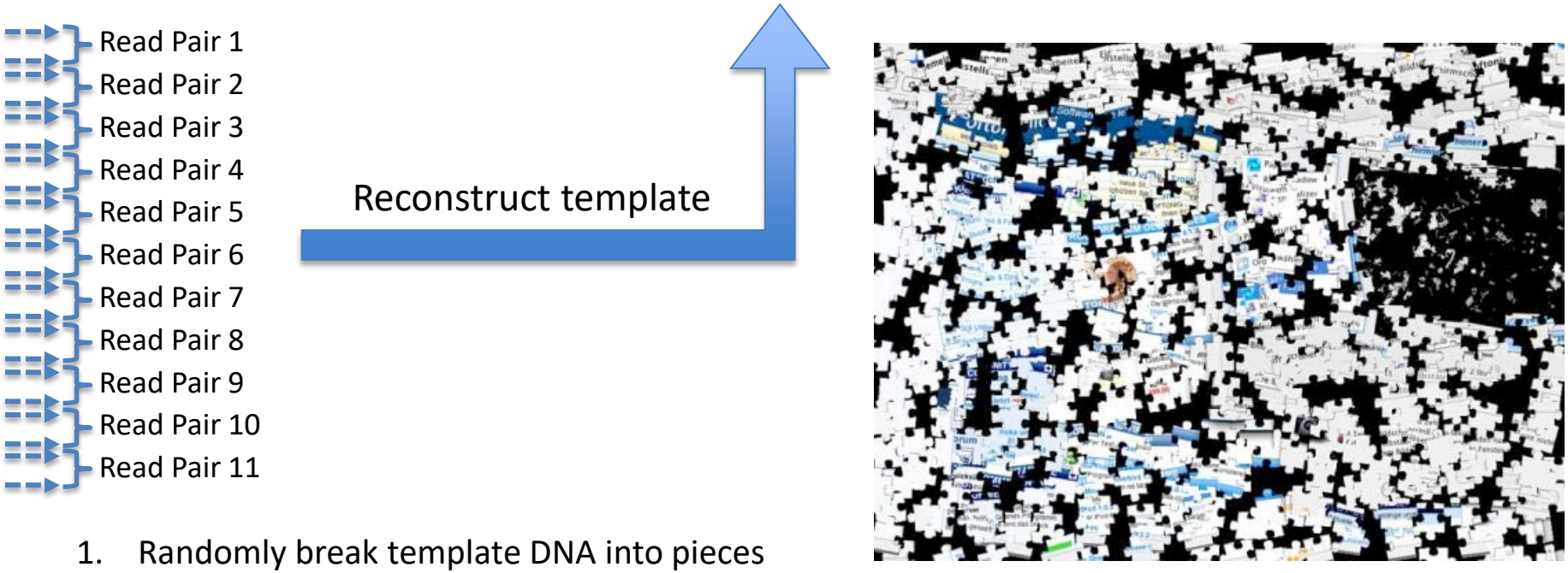
Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. Langmead et al. Genome Biology (2009)
doi:10.1186/gb-2009-10-3-r25 (BOWTIE)

Mapping Reads on a Genomic Sequence: An Algorithmic Overview and a Practical Comparative Analysis
Schbath et al. Journal of computational biology (2012)
doi:10.1089/cmb.2012.0022 (Review)

NextGenMap: fast and accurate read mapping in highly polymorphic genomes. Sedlazeck et al. Bioinformatics (2013)
doi:10.1093/bioinformatics/btt468 (NextGenMap)

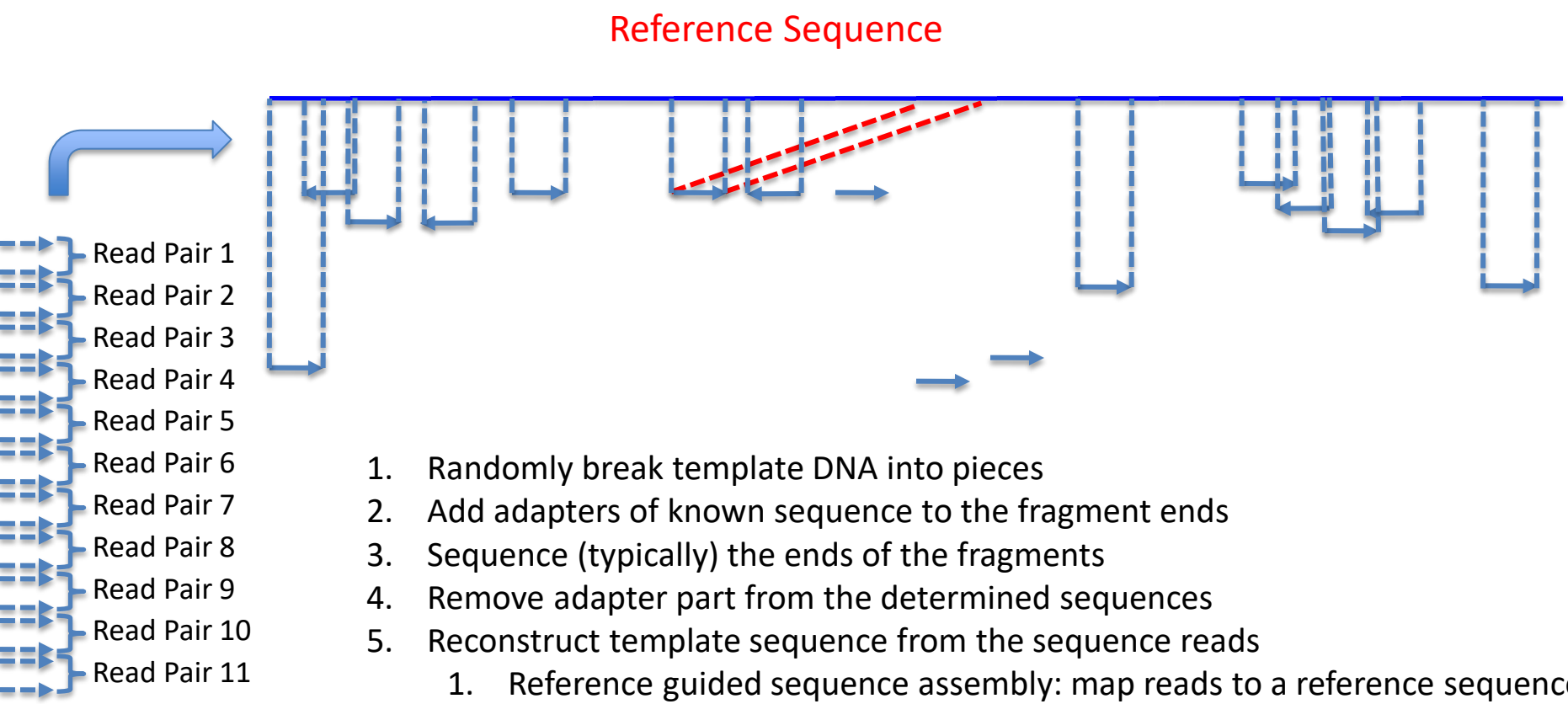
REFERENCE BASED MAPPING

Strategies to sequence long DNA molecules: Shotgun sequencing



1. Randomly break template DNA into pieces
2. Add adapters of known sequence to the fragment ends
3. Sequence (typically) the ends of the fragments
4. Identify and remove adapter part from the determined sequences
5. Reconstruct template sequence from the sequence reads

Strategies to sequence long DNA molecules: Shotgun sequencing and **reference guided** sequence assembly

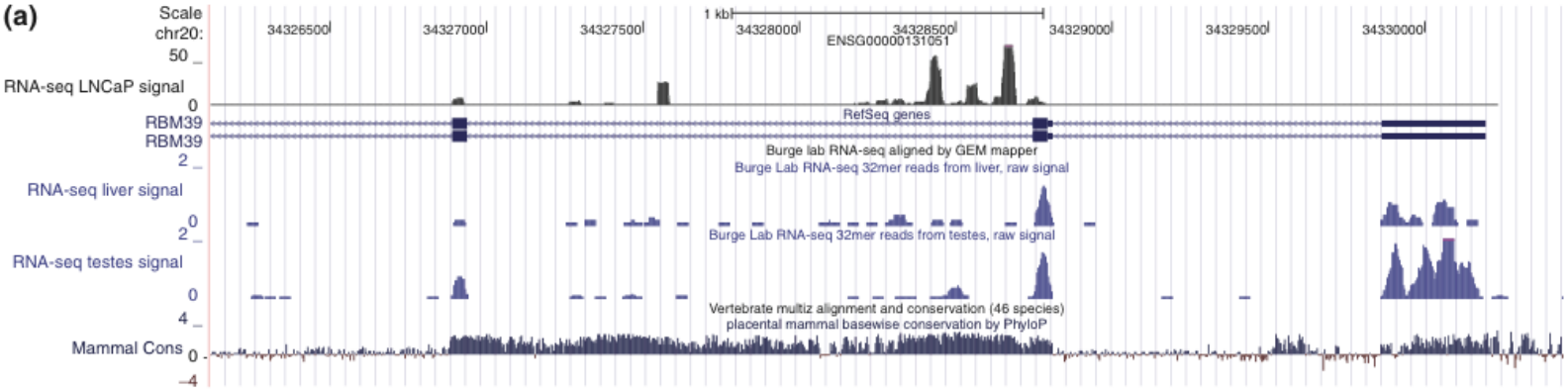


1. Randomly break template DNA into pieces
2. Add adapters of known sequence to the fragment ends
3. Sequence (typically) the ends of the fragments
4. Remove adapter part from the determined sequences
5. Reconstruct template sequence from the sequence reads
    1. Reference guided sequence assembly: map reads to a reference sequence. Note that reads can map equally good to more than one position in the reference genome, e.g. due to repeats in the reference.

Reference sequence

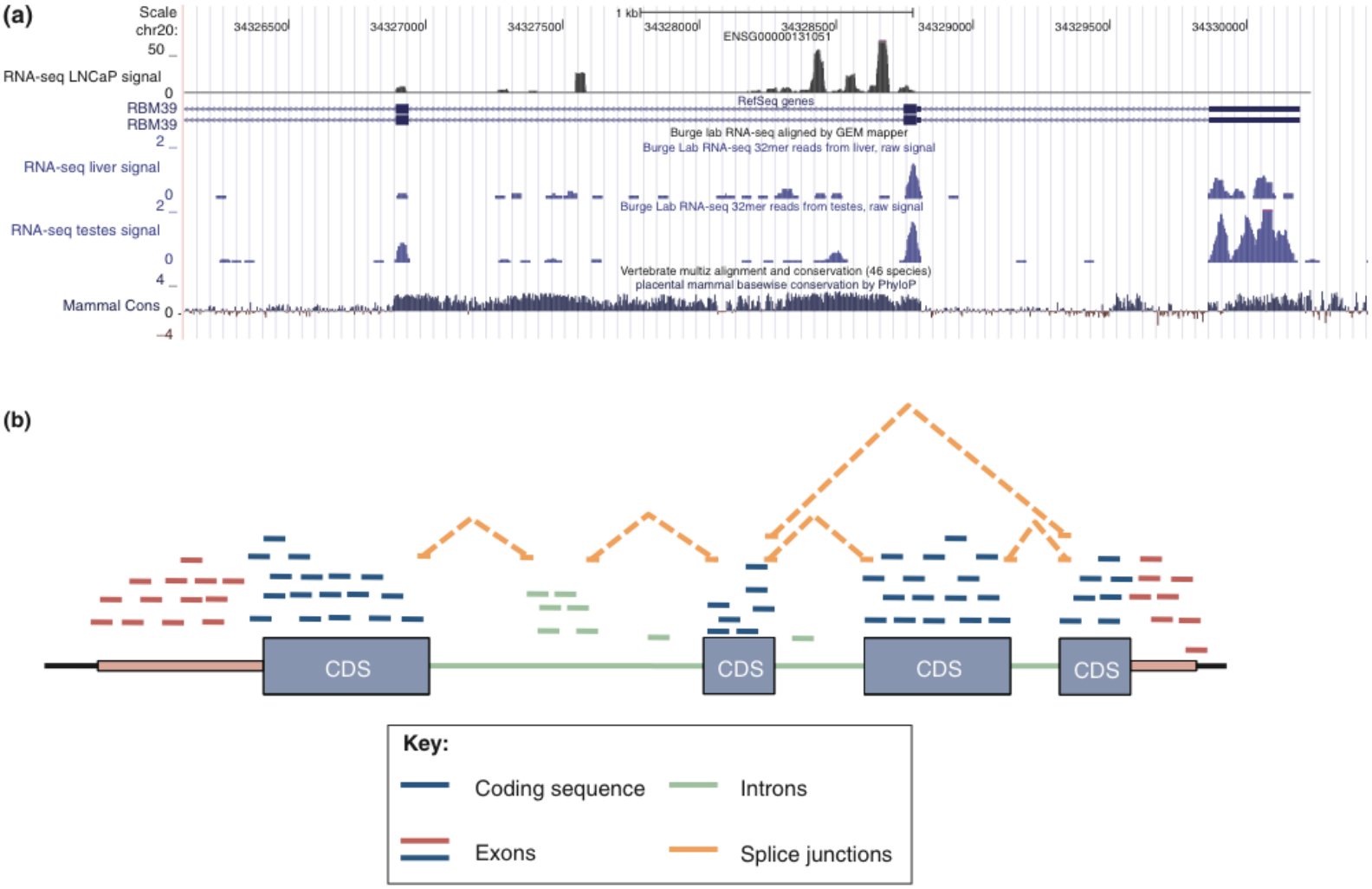
a) e.g. genome of a different individual from the same species to study species diversity
b) e.g. genome of a closely related species

Short Read Application: RNA seq mapping



from Oshlack et al. *Genome Biology* 2010, **11**:220

RNA-seq mapping helps building hypotheses concerning gene structure



from Oshlack et al. *Genome Biology* 2010, **11**:220

Short Read Applications

- Genotyping

Goal: identify variations

```
                                                GGTATAC...
...CCATAG    TATGCGCCC    CGGAAATTTCGGTATAC
...CCAT   CTATATGCG        TCGGAAATT  CGGTATAC
...CCATGGCTATATG      CTATCGGAAA   GCGGTATA
...CCAAGGCTATAT      CCTATCGGA   TTGCGGTA  C...
...CCAAGGCTATAT  GCCCTATCG     TTTGCGGT   C...
...CC AGGCTATAT  GCCCTATCGAAATTTGC    ATAC...
...CCTAGGCTATAGCGCCCTA    AAATTTGC GTATAC...
...CCATAGGCTATATGCGCCCTATCGGCAATTTGCGGTA
TAC...
```

Goal: classify, measure significant peaks

- RNA-seq, ChIP-seq, Methyl-seq, Ribo-seq

```
                    GAAATTTGC
                  GGAAATTTG
               CGGAAATTT
               CGGAAATTT
              TCGGAAATT
        CTATCGGAAA
     CCTATCGGA          TTTGCGGT
GCCCTATCG        AAATTTGC
...CC  GCCCTATCG        AAATTTGC        ATAC...
...CCATAGGCTATATGCGCCCTATCGGCAATTTGCGGTATAC...
```

Challenges

- mapping millions/billions of reads to a large genome is hard:
  - how quickly can we map the reads to the genome?
  - how do we deal with multiple mapping positions?
  - how do we deal with sequencing errors and genetic divergence/diversity
  - how do we deal with reads that span intron-exon boundaries?

SHORT READ ALIGNMENT

Short Read Alignment

- Given a reference and a set of reads, report at least one "good" local alignment for each read if one exists
  - Approximate answer to: from where in genome did the read originate?

What is "good"?

- Fewer mismatches is better
- Failing to align a low-quality base is better than failing to align a high-quality base

```
…TGATCATA…   better than   …TGATCATA…
 GATCAA                     GAGAAT

…TGATATTA…   better than   …TGATcaTA…
 GATca                      GTACAT
```

Finding mapping positions is, in principle, very easy!

Genome to search in:

AATGAGACATGAA

Reads to search:

```
Query1: CATG
Query2: ATGT
```

Finding mapping positions is, in principle, very easy.

Genome: AATGAGACATGAA

Query1: CATG

Naïve approach

```
            1234567891234
Searchstring: AATGAGACATGAA
              CATG
              ✖+++
```

Just slide the word along the sequence and stop when either end of sequence is reached or mapping position is found.

Finding mapping positions is, in principle, very easy.

Genome: AATGAGACATGAA

Query1: CATG

Naïve approach

```
1234567891234
Searchstring: AATGAGACATGAA
              CATG
              ✖✖✖✖
```

Just slide the word along the sequence and stop when either end of sequence is reached or mapping position is found.

Finding mapping positions is, in principle, very easy.

Genome: AATGAGACATGAA

Query1: CATG

Naïve approach

1234567891234
Searchstring: AATGAGACATGAA
CATG

✖✖✖✚

Just slide the word along the sequence and stop when either end of sequence is reached or mapping position is found.

Finding mapping positions is, in principle, very easy.

Genome: AATGAGACATGAA

Query1: CATG

Naïve approach

1234567891234
Searchstring: AATGAGACATGAA
CATG
++++

Query1 maps to position 8-12 in Genome

Full match found,
Output result

Finding mapping positions is, in principle, very easy.

Genome: AATGAGACATGAA

Query2: TTGT

Naïve approach

```
1234567891234
Searchstring: AATGAGACATGAA
              TTGT
              ✖✖✖✖
```

At most n-k comparisons, with n is the length of the search string, and k is the query length (read length).
This is not feasible for short read mapping.

Run Time of the Naïve approach

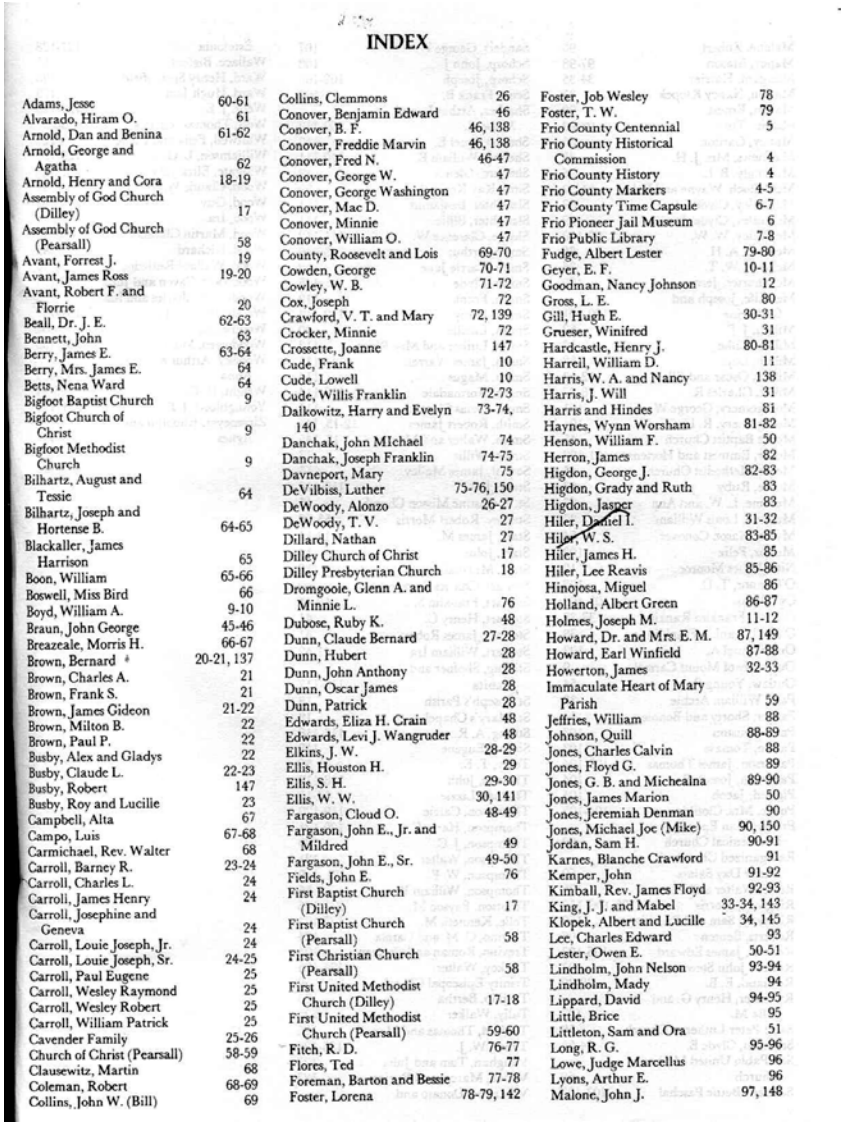
Naive approach:

- $O(L_G\ L_r\ N_r)$
- L_G → Size of the genome sequence
- L_r → Size of the read
- N_r → Number of the reads

Allowing gaps:

- Needleman Wunsch as dynamic programming algorithm
- Same complexity $O(L_G\ L_r\ N_r)$

Indexing speeds up searches



INDEX

Adams, Jesse 60-61
Alvarado, Hiram O. 61
Arnold, Dan and Benina 61-62
Arnold, George and Agatha 62
Arnold, Henry and Cora 18-19
Assembly of God Church (Dilley) 17
Assembly of God Church (Pearsall) 58
Avant, Forrest J. 19
Avant, James Ross 19-20
Avant, Robert F. and Florrie 20
Beall, Dr. J. E. 62-63
Bennett, John 63
Berry, James E. 63-64
Berry, Mrs. James E. 64
Betts, Nena Ward 64
Bigfoot Baptist Church 9
Bigfoot Church of Christ 9
Bigfoot Methodist Church 9
Bilhartz, August and Tessie 64
Bilhartz, Joseph and Hortense B. 64-65
Blackaller, James Harrison 65
Boon, William 65-66
Boswell, Miss Bird 66
Boyd, William A. 9-10
Braun, John George 45-46
Breazeale, Morris H. 66-67
Brown, Bernard 20-21, 137
Brown, Charles A. 21
Brown, Frank S. 21
Brown, James Gideon 21-22
Brown, Milton B. 22
Brown, Paul P. 22
Busby, Alex and Gladys 22
Busby, Claude L. 22-23
Busby, Robert 147
Busby, Roy and Lucille 23
Campbell, Alta 67
Campo, Luis 67-68
Carmichael, Rev. Walter 68
Carroll, Barney R. 23-24
Carroll, Charles L. 24
Carroll, James Henry 24
Carroll, Josephine and Geneva 24
Carroll, Louie Joseph, Jr. 24
Carroll, Louie Joseph, Sr. 24-25
Carroll, Paul Eugene 25
Carroll, Wesley Raymond 25
Carroll, Wesley Robert 25
Carroll, William Patrick 25
Cavender Family 25-26
Church of Christ (Pearsall) 58-59
Clausewitz, Martin 68
Coleman, Robert 68-69
Collins, John W. (Bill) 69
Collins, Clemmons 26
Conover, Benjamin Edward 46
Conover, B. F. 46, 138
Conover, Freddie Marvin 46, 138
Conover, Fred N. 46-47
Conover, George W. 47
Conover, George Washington 47
Conover, Mac D. 47
Conover, Minnie 47
Conover, William O. 47
County, Roosevelt and Lois 69-70
Cowden, George 70-71
Cowley, W. B. 71-72
Cox, Joseph 72
Crawford, V. T. and Mary 72, 139
Crocker, Minnie 72
Crossette, Joanne 147
Cude, Frank 10
Cude, Lowell 10
Cude, Willis Franklin 72-73
Dalkowitz, Harry and Evelyn 73-74, 140
Danchak, John MIchael 74
Danchak, Joseph Franklin 74-75
Davneport, Mary 75
DeVilbiss, Luther 75-76, 150
DeWoody, Alonzo 26-27
DeWoody, T. V. 27
Dillard, Nathan 27
Dilley Church of Christ 17
Dilley Presbyterian Church 18
Dromgoole, Glenn A. and Minnie L. 76
Dubose, Ruby K. 48
Dunn, Claude Bernard 27-28
Dunn, Hubert 28
Dunn, John Anthony 28
Dunn, Oscar James 28
Dunn, Patrick 28
Edwards, Eliza H. Crain 48
Edwards, Levi J. Wangruder 48
Elkins, J. W. 28-29
Ellis, Houston H. 29
Ellis, S. H. 29-30
Ellis, W. W. 30, 141
Fargason, Cloud O. 48-49
Fargason, John E., Jr. and Mildred 49
Fargason, John E., Sr. 49-50
Fields, John E. 76
First Baptist Church (Dilley) 17
First Baptist Church (Pearsall) 58
First Christian Church (Pearsall) 58
First United Methodist Church (Dilley) 17-18
First United Methodist Church (Pearsall) 59-60
Fitch, R. D. 76-77
Flores, Ted 77
Foreman, Barton and Bessie 77-78
Foster, Lorena 78-79, 142
Foster, Job Wesley 78
Foster, T. W. 79
Frio County Centennial 5
Frio County Historical Commission 4
Frio County History 4
Frio County Markers 4-5
Frio County Time Capsule 6-7
Frio Pioneer Jail Museum 6
Frio Public Library 7-8
Fudge, Albert Lester 79-80
Geyer, E. F. 10-11
Goodman, Nancy Johnson 12
Gross, L. E. 80
Gill, Hugh E. 30-31
Grueser, Winifred 31
Hardcastle, Henry J. 80-81
Harreil, William D. 11
Harris, W. A. and Nancy 138
Harris, J. Will 31
Harris and Hindes 81
Haynes, Wynn Worsham 81-82
Henson, William F. 50
Herron, James 82
Higdon, George J. 82-83
Higdon, Grady and Ruth 83
Higdon, Jasper 83
Hiler, Daniel I. 31-32
Hiler, W. S. 83-85
Hiler, James H. 85
Hiler, Lee Reavis 85-86
Hinojosa, Miguel 86
Holland, Albert Green 86-87
Holmes, Joseph M. 11-12
Howard, Dr. and Mrs. E. M. 87, 149
Howard, Earl Winfield 87-88
Howerton, James 32-33
Immaculate Heart of Mary Parish 59
Jeffries, William 88
Johnson, Quill 88-89
Jones, Charles Calvin 88
Jones, Floyd G. 89
Jones, G. B. and Michealna 89-90
Jones, James Marion 50
Jones, Jeremiah Denman 90
Jones, Michael Joe (Mike) 90, 150
Jordan, Sam H. 90-91
Karnes, Blanche Crawford 91
Kemper, John 91-92
Kimball, Rev. James Floyd 92-93
King, J. J. and Mabel 33-34, 143
Klopek, Albert and Lucille 34, 145
Lee, Charles Edward 93
Lester, Owen E. 50-51
Lindholm, John Nelson 93-94
Lindholm, Mady 94
Lippard, David 94-95
Little, Brice 95
Littleton, Sam and Ora 51
Long, R. G. 95-96
Lowe, Judge Marcellus 96
Lyons, Arthur E. 96
Malone, John J. 97, 148

Indexing:

- Allow targeted search
- Genome distributed in chapter / keywords

Indexing speeds up searches

Searchstring: CATGAGACATGAA

Query1: GAGA
Query2: CATG
Query3: ATGT

1) Decide on a word length *k*, e.g., k=3
2) Build hash table from search string, storing every word occurring in S together with its start position.
3) Process query and search for each word occurring in Q1 whether it is in the hash table.
4) Repeat for Q2.

Kmer	Position
CAT	1,8
ATG	2,9
TGA	3,10
GAG	4
AGA	5
GAC	6
ACA	7
GAA	11

Q1.1

Kmer	Position
GAG	1
AGA	2

Q1.2

Two lookups are sufficient to find *Q1* in *S*

Indexing handling mismatches?

2) How does the mapper deal with queries that 'almost' match the reference?

Q3 matches the reference with one mismatch

Relevant for sensitivity and specificity of the mapping. Allowing more mismatches increases sensitivity (consider sequencing error and genetic diversity) but decreases specificity (more false positives).

Kmer	Position
CAT	1,8
ATG	2,9
TGA	3,10
GAG	4
AGA	5
GAC	6
ACA	7
GAA	11

Q3.1

Kmer	Position
ATG	1
TGT	2

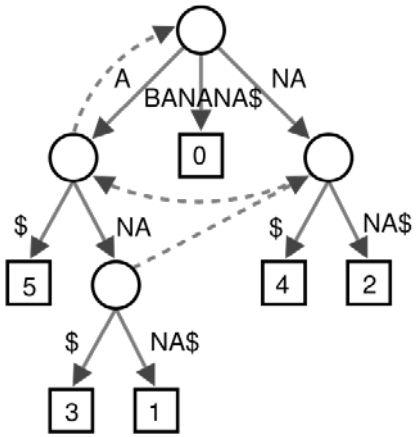
One mismatch

The 2nd lookup indicates that *Q3* is almost in *S*

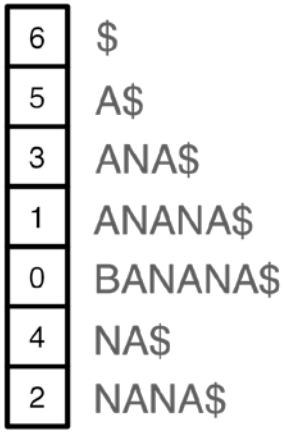
Main differences between mapping approaches

3) What kind of index does the mapper use?

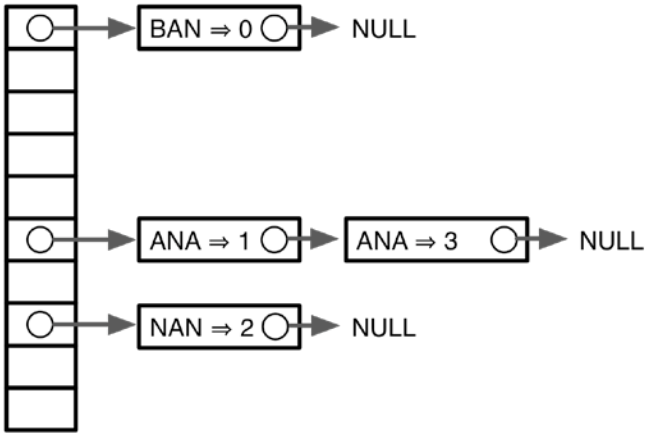
Relevant for speed and memory footprint of the mapper



Suffix tree



Suffix array



Seed hash tables

Many variants, incl. spaced seeds

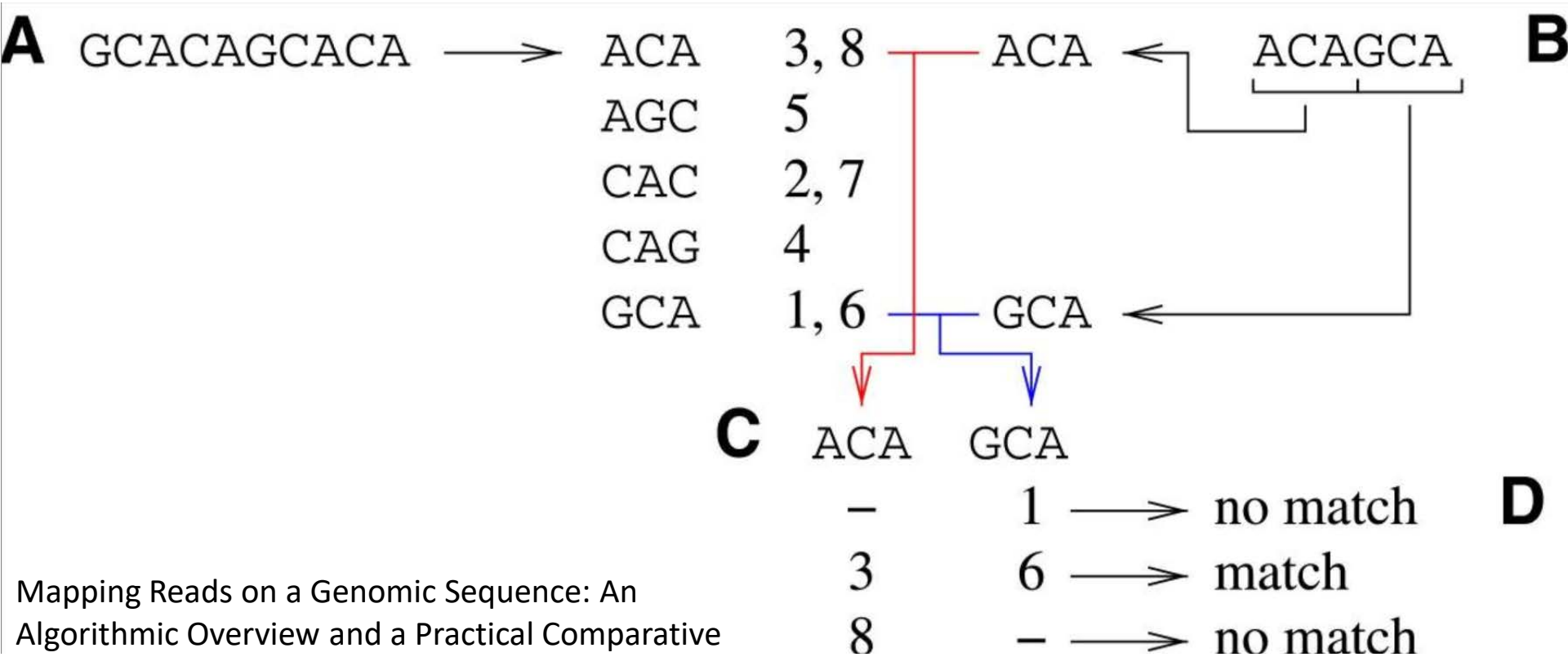
$BANANA
A$BANAN
ANA$BAN
ANANA$B
BANANA$
NA$BANA
NANA$BA

Burrows Wheeler Transformation

HASH-BASED MAPPING

The "traditional way": Hash tables

- Used by MAQ, Eland, SOAP, SHRiMP, ZOOM, partially by Mosaik, SSAHA2, Stampy

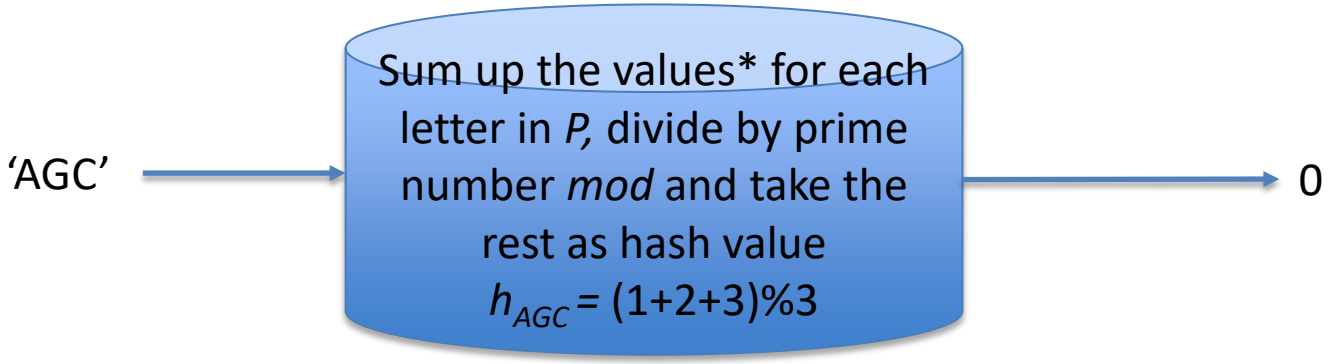


Mapping Reads on a Genomic Sequence: An Algorithmic Overview and a Practical Comparative Analysis (2012) Sophie Schbath

The use of 'hashing' in exact pattern search (Rabin-Karp; O(n+m))

Approach:

1) Use a 'hash-function' to transform pattern *P* into a numerical hash value h_P.

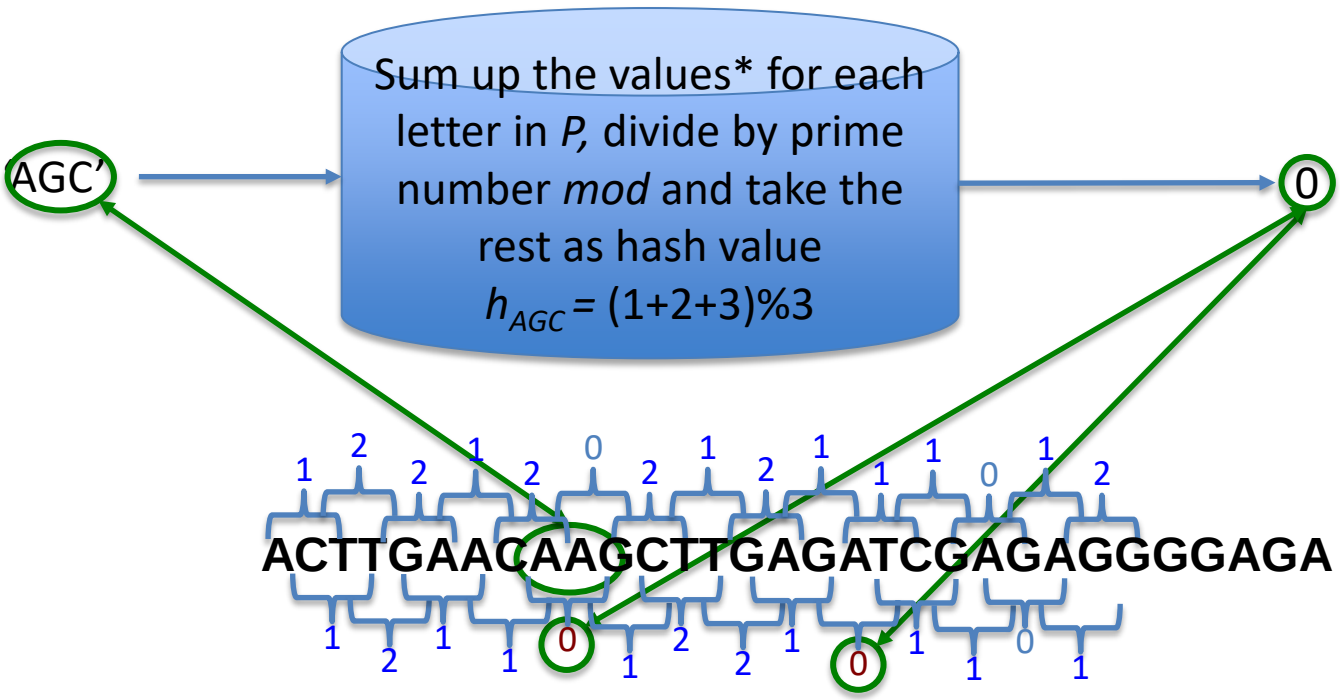


*JUST AN EXAMPLE: *v*(A)=1, *v*(C)=2, *v*(G)=3, *v*(T)=4; *mod*=3

The use of 'hashing' in exact pattern search (Rabin-Karp; O(n+m))

Approach:

1) Use a 'hash-function' to transform pattern *P* into a numerical hash value h_P.



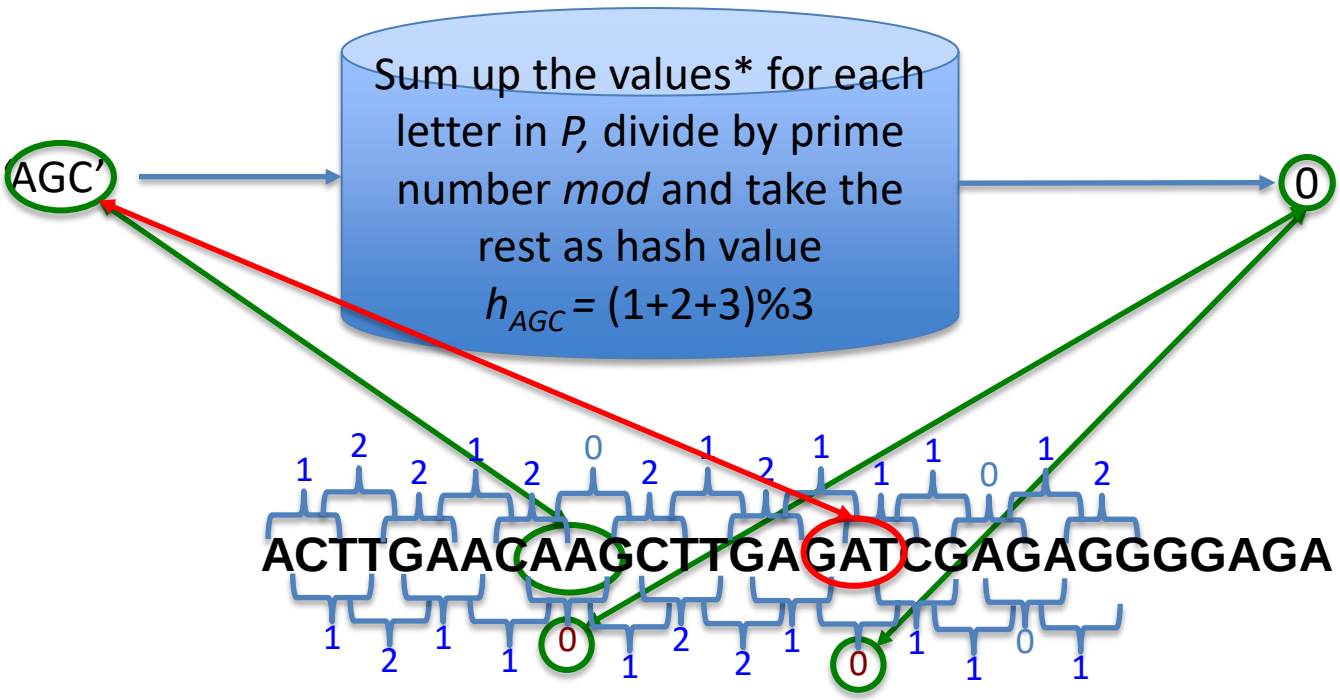
2) Search the text *T* starting from left for words of length $|P|$ having the same hash value as *P*.

*JUST AN EXAMPLE: *v*(A)=1, *v*(C)=2, *v*(G)=3, *v*(T)=4; *mod*=3

The use of 'hashing' in exact pattern search (Rabin-Karp; O(n+m))

Approach:

1) Use a 'hash-function' to transform pattern *P* into a numerical hash value h_P.



2) Search the text *T* starting from left for words of length $|P|$ having the same hash value as *P*.

*JUST AN EXAMPLE: *v*(A)=1, *v*(C)=2, *v*(G)=3, *v*(T)=4; *mod*=3

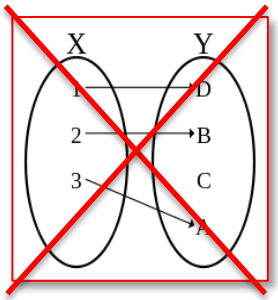
The use of 'hashing' in exact pattern search (Rabin-Karp; O(n+m))

Approach:

1) Use a 'hash-function' to transform pattern *P* into a numerical hash value h_P.

2) Search the text *T* starting from left for words of length $|P|$ having the same hash value as *P*.

3) Given a word *K* with $h_k = h_P$ was found, perform an exact string comparison to verify that *K* == *P.* (Note, the projection of words with length $|P|$ in the space of hash values is **not** injective (linkseindeutig!).

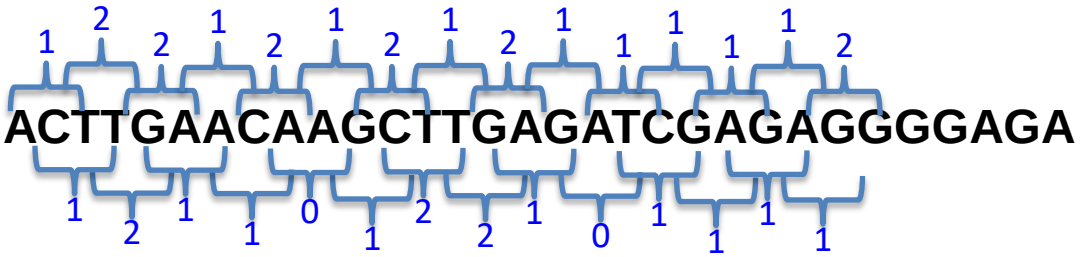


*JUST AN EXAMPLE: *v*(A)=1, *v*(C)=2, *v*(G)=3, *v*(T)=4; *mod*=3

‘hashing’ in combination with hash tables help to reduce the average time complexity of the pattern search to O(1), i.e. constant in time*

Idea: Speed up pattern search by creating look-up tables storing the hash values

1) Search the text *T* starting from left for words of length *k* and compute their hash value h_k



2) Store the hash values together with the starting point of the corresponding words in a hash table. Note, a hash table is nothing but a special way of indexing your data, just like a phone book. This will provide direct access to your potential matches in the pattern search once you know the hash value of *P*.

Hash value	Position in string
0	10,11,20,25,26
1	1,2,6,7,8, 12,15,18,19,21,22,23,24,27,28
2	3,4,5,9,13,14,16,17,29

*note that this ignores the time and space you need to populate your hash table!

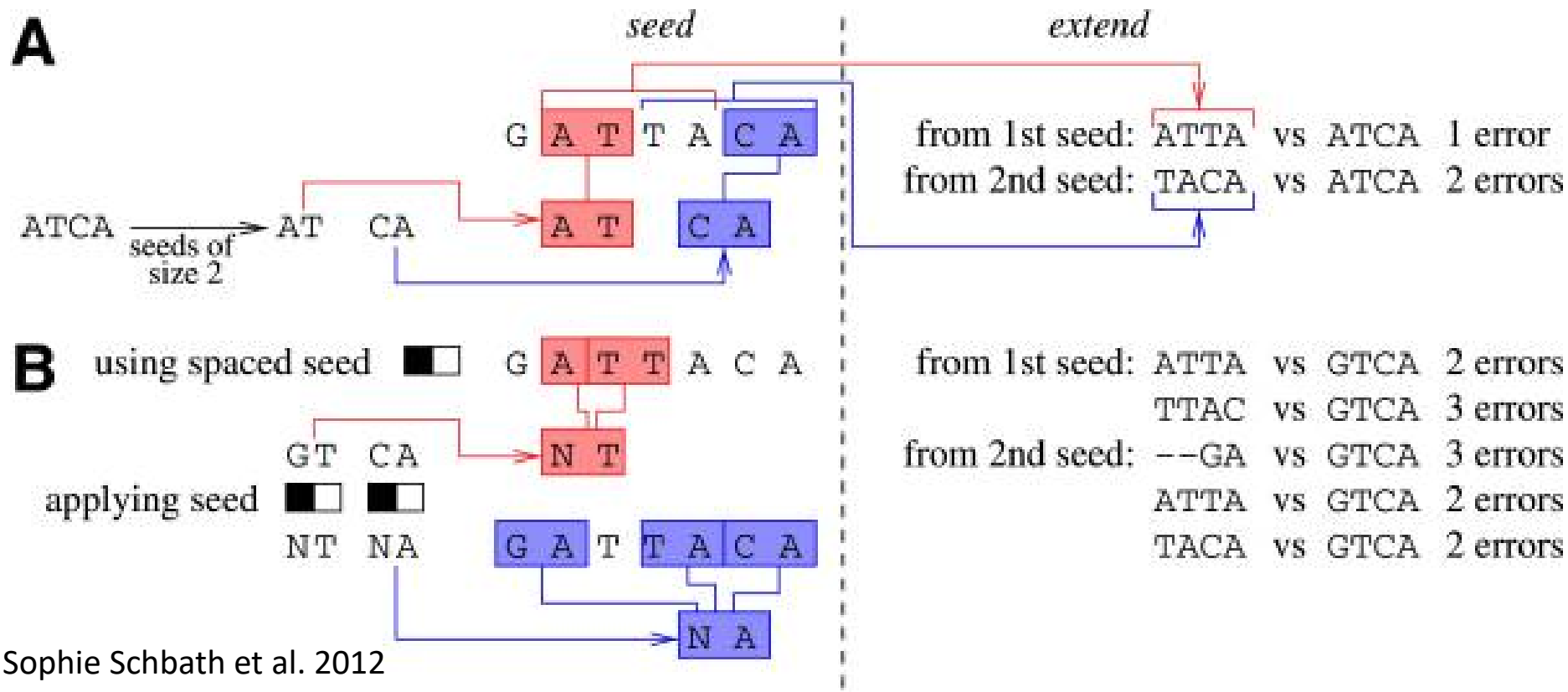
Seed and extend with **local alignment**

- SSAHA & Stampy:
  - Use k-mer (shorter than read) to find it in the genome
  - Seed regions will be extended by Needleman-Wunsch

- Drawback:
  - Many regions have to be analyzed in the extend phase

Seed and Extend the **pigeon hole principle**

- MAQ, SOAP, RMAP:
  - Chop read in k-mers (allowing errors)
  - All k-mers in the genome + correct order + adjacent to each other → read found



Sophie Schbath et al. 2012

Seed and Extend the **q-gram filtering**

- SHRiMP2 & RazerS:
  - Chop read in k-mers but overlapping
  - If enough k-mers map in a small region a more careful alignment will be done

- Drawback:
  - List to large to be kept in memory
  - Characters stored in 8 bits (2 bits per Nucleotide) but ambiguity code has to be overcome

Why does MAQ* use **pigeon hole principle** and **spaced seeds** for mapping?

Issues to solve by seed and extend:

- Shorter seeds map more regions on the genome
- Minimum of 10 nucleotides per k-mer
- Allowing "Don't care" positions to be able to find seeds → **spaced seed approach**

Reference

AGACTGAGGTACGTAGACCATGATCGATACCCAAAAAGCTAGA

GTACGTAGACGATGATCCATACCCAAAA

Read (28 bp prefix)

MAQ uses seed pairs as it allows, per default no more than 2 mismatches between seed and reference. As each 28 mere is represented by 4 non-overlapping seeds, we **always have at least 2 seeds** that must result in a perfect match to the reference.

MAQ*: **pigeon hole principle** for mapping

Example: read1 - gatgtgacatacctgttctactgaggct

Build **six** hash tables (templates) for the reads (only first 28 bp are considered) **only** from the colored nucleotides

6 'Templates'

	HASH	
gatgtgacatacctgttctactgaggct	→	2057673064
gatgtgacatacctgttctactgaggct	→	3178370917
gatgtgacatacctgttctactgaggct	→	773088662
gatgtgacatacctgttctactgaggct	→	1856750201
gatgtgacatacctgttctactgaggct	→	2510061809
gatgtgacatacctgttctactgaggct	→	119777054

hash value for template 1: 832589471

GENOME
ttcagttatccagaaaatgctattttccaaaatgaaatctaaaatagtaactcaagtgaaacattgtcagg
tgtgtaaggaaggaaaatatgaagggctacccacaaacccacaaacaacggaaactccaattcctaagtt
tccgggacacactattcatatagatataatttctacagacaaaaacgggtacttacggcaattcacaaattttc
aaaacttgcgaaagcaaaaataaaaaaaattcaaaatcaatagaagacataagaaaacctttacatgac
atcttattttattttggagtaccaaaatacgttgtaatggatatagaaaaatcctttaattccgcaacaacagcctttt
atgatgaaagaccagctgggcatacaaattttcaaagcatccccttataaaagttctgtaaacggacaaata
gaacggtttcattctatcctcgctgaaattaaaagatgtttaaaaactaaacaggtacaccgaacatttgaaga
acagttcaattcagctgtctaggaatataactacacaattcaccctgtatcaaaaatacaaaacaaaataac
gccccttaaaaatatttttggcagaaatataaccactgatccaggaaaatatgacgaaagcagataaggcaa
catcgaaaacctataatcaaaacaggcaacaggcttaaaaaccataacgaaaaaagcaataagatcaa
agattatgagccaggacaaacagtttttataaagcaaatcacaaggccaggttctaagctgtacaagaaag
aaacattaaaagaaaacagacaaacatttgttatcacagaagcaggaagatgtgacatacctgttctactg
aaggct

compute hash values for spaced seeds in reference (on both strands and for one of the six templates) and perform lookups in hash tables of the reads

*Li et al *Genome Res. 2008. 18: 1851-1858*

MAQ*: **pigeon hole principle** for mapping

Example: read1 - gatgtgacatacctgttctactgaggct

Build **six** hash tables for the reads (only first 28 bp) using **only** the colored nucleotides

gatgtgacatacctgttctactgaggct	HASH →	2057673064
gatgtgacatacctgttctactgaggct	→	3178370917
gatgtgacatacctgttctactgaggct	→	773088662
gatgtgacatacctgttctactgaggct	→	1856750201
gatgtgacatacctgttctactgaggct	→	2510061809
gatgtgacatacctgttctactgaggct	→	119777054

1258119214

GENOME

ttcagttatccagaaaatgctatttcccaaaatgaaatctaaaatagtaactcaagtgaaacattgtcagg
tgtgtaaggaaggaaaatatgaagggctacccacaaacccacaaacaacggaaactccaattcctaagtt
tccgggacacactattcatatagatataatttctacagacaaaaacgggtacttacggcaattcacaaattttc
aaaacttgcgaaagcaaaaataaaaaaaattcaaaatcaatagaagacataagaaaacctttacatgac
atcttattttattttggagtaccaaaatacgttgtaatggatatagaaaaatcctttaattccgcaacaacagcctttt
atgatgaaagaccagctgggcatacaaattttcaaagcatccccttataaaagttctgtaaacggacaaata
gaacggtttcattctatcctcgctgaaattaaaagatgtttaaaaactaaacaggtacaccgaacatttgaaga
acagttcaattcagctgtctaggaatataactacacaattcaccctgtatcaaaaatacaaaacaaaataac
gcccttaaaaatatttttggcagaaatataaccactgatccaggaaaatatgacgaaagcagataaggcaa
catcgaaaacctataatcaaaacaggcaacaggcttaaaaaccataacgaaaaaagcaataagatcaa
agattatgagccaggacaaacagtttttataaagcaaatcacaaggccaggttctaagctgtacaagaaag
aaacattaaaagaaaacagacaaacatttgttatcacagaagcaggaagatgtgacatacctgttctactg
aaggct

continue with next word in the reference from the same template….
until the entire reference sequence has been used.

*Li et al *Genome Res. 2008. 18: 1851-1858*

MAQ *: **pigeon hole principle** for mapping

Example: read1 - gatgtgacatacctgttctactgaggct

Build **six** hash tables for the reads (only first 28 bp) using **only** the colored nucleotides

gatgtgacatacctgttctactgaggct	HASH →	2057673064
gatgtgacatacctgttctactgaggct	→	3178370917
gatgtgacatacctgttctactgaggct	→	773088662
gatgtgacatacctgttctactgaggct	→	1856750201
gatgtgacatacctgttctactgaggct	→	2510061809
gatgtgacatacctgttctactgaggct	→	119777054

HIT: Calculate the sum of qualities of mismatched bases *q* over the whole length of the read and store together with hit position.

For each read, MAQ stores score and position of only the 2 best hits and the number of 0-, 1-, and 2-mismatch seed positions

GENOME
ttcagttatccagaaaatgctattttcccaaaatgaaatctaaaatagtaactcaagtgaaacattgtcaggtgt
gtaaggaaggaaaatatgaagggctacccacaaacccacaaacaacggaaactccaattcctaagtttcc
gggacacactattcatatagatataatttctacagacaaaaacgggtacttacggcaattcacaaattttcaaa
acttgcgaaagcaaaaataaaaaaaattcaaaatcaatagaagacataagaaaacctttacatgacatctt
attttattttggagtaccaaaatacgttgtaatggatatagaaaaatcctttaattccgcaacaacagcctttatg
atgaaagaccagctgggcatacaaattttcaaagcatccccttataaaagttctgtaaacggacaaatagaa
cggtttcattctatcctcgctgaaattaaaagatgtttaaaaactaaacaggtacaccgaacatttgaagaaca
gttcaattcagctgtctaggaatataactacacaattcaccctgtatcaaaaatacaaaacaaaataacgccc
ttaaaaatatttttggcagaaatataaccactgatccaggaaaatatgacgaaagcagataaggcaacatc
gaaaacctataatcaaaacaggcaacaggcttaaaaaccataacgaaaaaagcaataagatcaaagat
tatgagccaggacaaacagtttttataaagcaaatcacaaggccaggttctaagctgtacaagaaagaaac
attaaaagaaaacagacaaacatttgttatcacagaagcaggaa**gatgtgacatacctgttctactgaa**
ggct

2057673064

*Li et al *Genome Res. 2008. 18: 1851-1858*

MAQ* uses **spaced seeds** for mapping

Example: read1 - gatgtgacatacctgttctactgaggct

Build **six** hash tables for the reads (only first 28 bp) using **only** the colored nucleotides

gatgtgacatacctgttctactgaggct	HASH →	2057673064
gatgtgacatacctgttctactgaggct	→	3178370917
gatgtgacatacctgttctactgaggct	→	773088662
gatgtgacatacctgttctactgaggct	→	1856750201
gatgtgacatacctgttctactgaggct	→	2510061809
gatgtgacatacctgttctactgaggct	→	119777054

hash value for template 2

GENOME

ttcagttatccagaaaatgctattttcccaaaatgaaatctaaaatagtaactcaagtgaaacattgtcagg
tgtgtaaggaaggaaaatatgaagggctacccacaaacccacaaacaacggaaactccaattcctaagtt
tccgggacacactattcatatagatataatttctacagacaaaaacgggtacttacggcaattcacaaattttc
aaaacttgcgaaagcaaaaataaaaaaaattcaaaatcaatagaagacataagaaaacctttacatgac
atcttattttattttggagtaccaaaatacgttgtaatggatatagaaaaatcctttaattccgcaacaacagcctttt
atgatgaaagaccagctgggcatacaaattttcaaagcatccccttataaaagttctgtaaacggacaaata
gaacggtttcattctatcctcgctgaaattaaaagatgtttaaaaactaaacaggtacaccgaacatttgaaga
acagttcaattcagctgtctaggaatataactacacaattcaccctgtatcaaaaatacaaaacaaaataac
gccccttaaaaatatttttggcagaaatataaccactgatccaggaaaatatgacgaaagcagataaggcaa
catcgaaaacctataatcaaaacaggcaacaggcttaaaaaccataacgaaaaaagcaataagatcaa
agattatgagccaggacaaacagtttttataaagcaaatcacaaggccaggttctaagctgtacaagaaag
aaacattaaaagaaaacagacaaacatttgttatcacagaagcaggaagatgtgacatacctgttctactg
aaggct

compute hash values for spaced seeds in reference (on both strands) and perform lookup in hash tables of the reads

Li et al *Genome Res. 2008. 18: 1851-1858*

MAQ* uses **spaced seeds** for mapping

Example: read1 - gatgtgacatacctgttctactgaggct

Build **six** hash tables for the reads (only first 28 bp) using **only** the colored nucleotides

gatgtgacatacctgttctactgaggct	HASH →	2057673064
gatgtgacatacctgttctactgaggct	→	3178370917
gatgtgacatacctgttctactgaggct	→	773088662
gatgtgacatacctgttctactgaggct	→	1856750201
gatgtgacatacctgttctactgaggct	→	2510061809
gatgtgacatacctgttctactgaggct	→	119777054

hash value 3

GENOME

ttcagttatccagaaaatgctattttcccaaaatgaaatctaaaatagtaactcaagtgaaacattgtcagg
tgtgtaaggaaggaaaatatgaagggctacccacaaacccacaaacaacggaaactccaattcctaagtt
tccgggacacactattcatatagatataatttctacagacaaaaacgggtacttacggcaattcacaaattttc
aaaacttgcgaaagcaaaaataaaaaaaattcaaaatcaatagaagacataagaaaacctttacatgac
atcttattttattttggagtaccaaaatacgttgtaatggatatagaaaaatcctttaattccgcaacaacagcctttt
atgatgaaagaccagctgggcatacaaattttcaaagcatccccttataaaagttctgtaaacggacaaata
gaacggtttcattctatcctcgctgaaattaaaagatgtttaaaaactaaacaggtacaccgaacatttgaaga
acagttcaattcagctgtctaggaatataactacacaattcaccctgtatcaaaaatacaaaacaaaataac
gccccttaaaaatatttttggcagaaatataaccactgatccaggaaaatatgacgaaagcagataaggcaa
catcgaaaacctataatcaaaacaggcaacaggcttaaaaaccataacgaaaaaagcaataagatcaa
agattatgagccaggacaaacagtttttataaagcaaatcacaaggccaggttctaagctgtacaagaaag
aaacattaaaagaaaacagacaaacatttgttatcacagaagcaggaagatgtgacatacctgttctactg
aaggct

compute hash values for spaced seeds in reference (on both strands) and perform lookup in hash tables of the reads

Li et al *Genome Res. 2008. 18: 1851-1858*

MAQ* uses **spaced seeds** for mapping

Example: read1 - gatgtgacatacctgttctactgaggct

Build **six** hash tables for the reads (only first 28 bp) using **only** the colored nucleotides

gatgtgacatacctgttctactgaggct	HASH →	2057673064
gatgtgacatacctgttctactgaggct	→	3178370917
gatgtgacatacctgttctactgaggct	→	773088662
gatgtgacatacctgttctactgaggct	→	1856750201
gatgtgacatacctgttctactgaggct	→	2510061809
gatgtgacatacctgttctactgaggct	→	119777054

hash value 4

GENOME

ttcagttatccagaaaatgctattttccaaaatgaaatctaaaatagtaactcaagtgaaacattgtcagg
tgtgtaaggaaggaaaatatgaagggctacccacaaacccacaaacaacggaaactccaattcctaagtt
tccgggacacactattcatatagatataatttctacagacaaaaacgggtacttacggcaattcacaaattttc
aaaacttgcgaaagcaaaaataaaaaaaattcaaaatcaatagaagacataagaaaacctttacatgac
atcttattttattttggagtaccaaaatacgttgtaatggatatagaaaaatcctttaattccgcaacaacagcctttt
atgatgaaagaccagctgggcatacaaattttcaaagcatccccttataaaagttctgtaaacggacaaata
gaacggtttcattctatcctcgctgaaattaaaagatgtttaaaaactaaacaggtacaccgaacatttgaaga
acagttcaattcagctgtctaggaatataactacacaattcaccctgtatcaaaaatacaaaacaaaataac
gcccttaaaaatatttttggcagaaatataaccactgatccaggaaaatatgacgaaagcagataaggcaa
catcgaaaacctataatcaaaacaggcaacaggcttaaaaaccataacgaaaaaagcaataagatcaa
agattatgagccaggacaaacagtttttataaagcaaatcacaaggccaggttctaagctgtacaagaaag
aaacattaaaagaaaacagacaaacatttgttatcacagaagcaggaagatgtgacatacctgttctactg
aaggct

compute hash values for spaced seeds in reference (on both strands) and perform lookup in hash tables of the reads

Li et al *Genome Res. 2008. 18: 1851-1858*

MAQ* uses **spaced seeds** for mapping

Example: read1 - gatgtgacatacctgttctactgaggct

Build **six** hash tables for the reads (only first 28 bp) using **only** the colored nucleotides

gatgtgacatacctgttctactgaggct	HASH →	2057673064
gatgtgacatacctgttctactgaggct	→	3178370917
gatgtgacatacctgttctactgaggct	→	773088662
gatgtgacatacctgttctactgaggct	→	1856750201
gatgtgacatacctgttctactgaggct	→	2510061809
gatgtgacatacctgttctactgaggct	→	119777054

hash value 5

GENOME

ttcagttatccagaaaatgctattttccaaaatgaaatctaaaatagtaactcaagtgaaacattgtcagg
tgtgtaaggaaggaaaatatgaagggctacccacaaacccacaaacaacggaaactccaattcctaagtt
tccgggacacactattcatatagatataatttctacagacaaaaacgggtacttacggcaattcacaaattttc
aaaacttgcgaaagcaaaaataaaaaaaattcaaaatcaatagaagacataagaaaacctttacatgac
atcttattttattttggagtaccaaaatacgttgtaatggatatagaaaaatcctttaattccgcaacaacagcctttt
atgatgaaagaccagctgggcatacaaattttcaaagcatccccttataaaagttctgtaaacggacaaata
gaacggtttcattctatcctcgctgaaattaaaagatgtttaaaaactaaacaggtacaccgaacatttgaaga
acagttcaattcagctgtctaggaatataactacacaattcaccctgtatcaaaaatacaaaacaaaataac
gcccttaaaaatatttttggcagaaatataaccactgatccaggaaaatatgacgaaagcagataaggcaa
catcgaaaacctataatcaaaacaggcaacaggcttaaaaaccataacgaaaaaagcaataagatcaa
agattatgagccaggacaaacagtttttataaagcaaatcacaaggccaggttctaagctgtacaagaaag
aaacattaaaagaaaacagacaaacatttgttatcacagaagcaggaagatgtgacatacctgttctactg
aaggct

compute hash values for spaced seeds in reference (on both strands) and perform lookup in hash tables of the reads

Li et al *Genome Res. 2008. 18: 1851-1858*

MAQ* uses **spaced seeds** for mapping

Example: read1 - gatgtgacatacctgttctactgaggct

Build **six** hash tables for the reads (only first 28 bp) using **only** the colored nucleotides

gatgtgacatacctgttctactgaggct	HASH →	2057673064
gatgtgacatacctgttctactgaggct	→	3178370917
gatgtgacatacctgttctactgaggct	→	773088662
gatgtgacatacctgttctactgaggct	→	1856750201
gatgtgacatacctgttctactgaggct	→	2510061809
gatgtgacatacctgttctactgaggct	→	119777054

hash value 6

GENOME

ttcagttatccagaaaatgctattttccccaaaatgaaatctaaaatagtaactcaagtgaaacattgtcagg
tgtgtaaggaaggaaaatatgaagggctacccacaaacccacaaacaacggaaactccaattcctaagtt
tccgggacacactattcatatagatataatttctacagacaaaaacgggtacttacggcaattcacaaattttc
aaaacttgcgaaagcaaaaataaaaaaaattcaaaatcaatagaagacataagaaaacctttacatgac
atcttattttattttggagtaccaaaatacgttgtaatggatatagaaaaatcctttaattccgcaacaacagcctttt
atgatgaaagaccagctgggcatacaaattttcaaagcatccccttataaaagttctgtaaacggacaaata
gaacggtttcattctatcctcgctgaaattaaaagatgtttaaaaactaaacaggtacaccgaacatttgaaga
acagttcaattcagctgtctaggaatataactacacaattcaccctgtatcaaaaatacaaaacaaaataac
gcccttaaaaatatttttggcagaaatataaccactgatccaggaaaatatgacgaaagcagataaggcaa
catcgaaaacctataatcaaaacaggcaacaggcttaaaaaccataacgaaaaaagcaataagatcaa
agattatgagccaggacaaacagtttttataaagcaaatcacaaggccaggttctaagctgtacaagaaag
aaacattaaaagaaaacagacaaacatttgttatcacagaagcaggaagatgtgacatacctgttctactg
aaggct

compute hash values for spaced seeds in reference (on both strands) and perform lookup in hash tables of the reads

Li et al *Genome Res. 2008. 18: 1851-1858*

MAQ: An overview

1. At the alignment stage, MAQ first searches for the ungapped match with lowest mismatch score, defined as the sum of qualities at mismatching bases.
2. MAQ only considers positions that have two or fewer mismatches in the first 28 bp (default parameters; speed-up).
3. Sequences that fail to reach a mismatch score threshold but whose mate pair is mapped are searched with a gapped alignment algorithm in the regions defined by the mate pair.
4. To evaluate the reliability of alignments, MAQ assigns each individual alignment a phred-scaled quality score (capped at 99), which measures the probability that the true alignment is not the one found by MAQ.
5. MAQ always reports a single alignment, and if a read can be aligned equally well to multiple positions, MAQ will randomly pick one position and give it a mapping quality zero. Because their mapping score is set to zero, reads that are mapped equally well to multiple positions will not contribute to variant calling.

NextGenMap

- Hash-based read mapper bridging:
  1. speed
  2. ability to map reads in highly polymorphic regions

- pitfalls:
  1. Mixing single end and paired end reads is not supported → corrupt the mapping results
  2. Not recommended to change parameters –kmer and kmer-skip
  3. Values between 0.5 and 0.8 give the best trade-off between speed and sensitivity

Algorithm Workflow

1. Indexing the reference genome (only once)

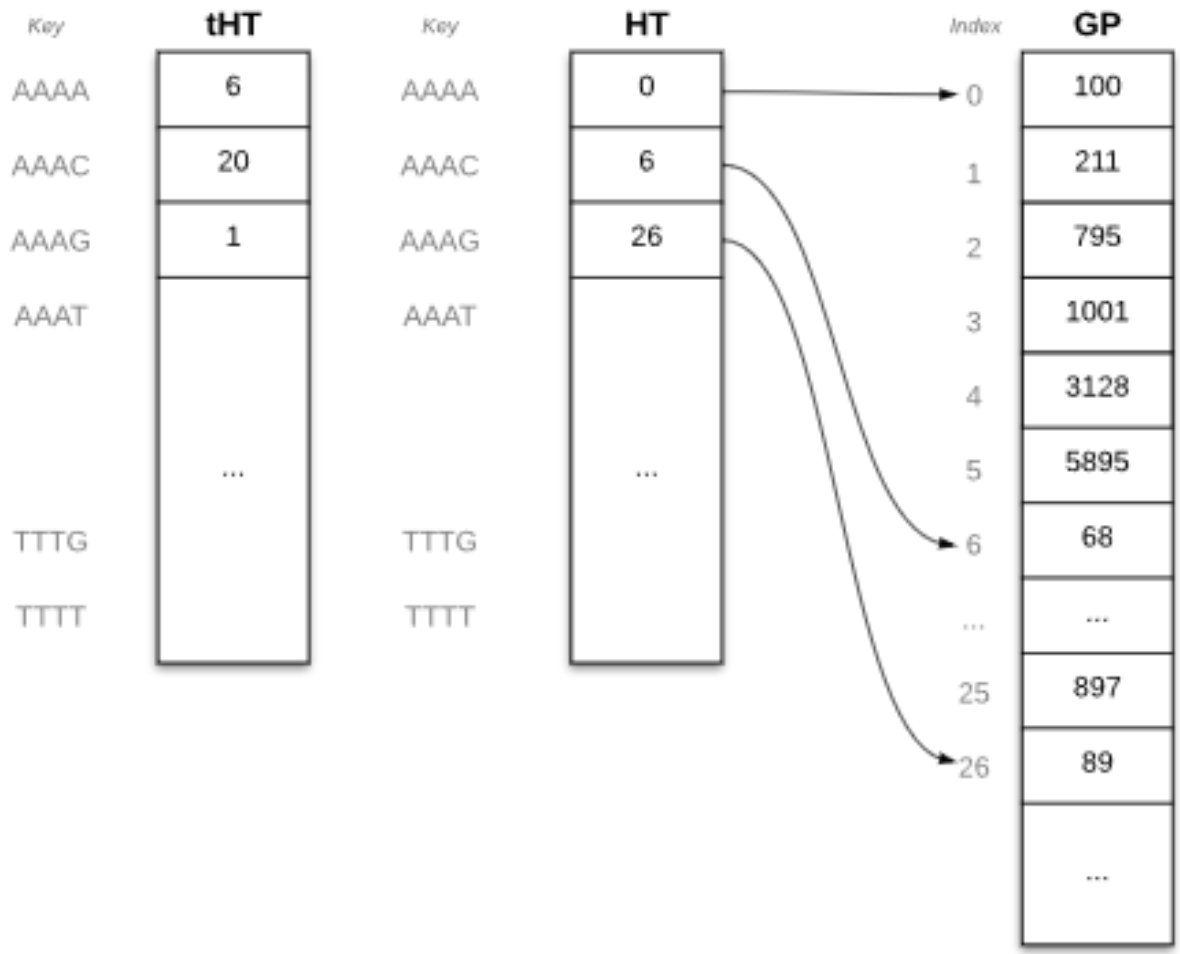
2. Identification of possible mapping regions on the reference genome (candidate mapping region (CMR) search)

3. Computation of alignment scores for all CMRs found in step 2

4. Computation of the full alignment for the best scoring CMR from step 3

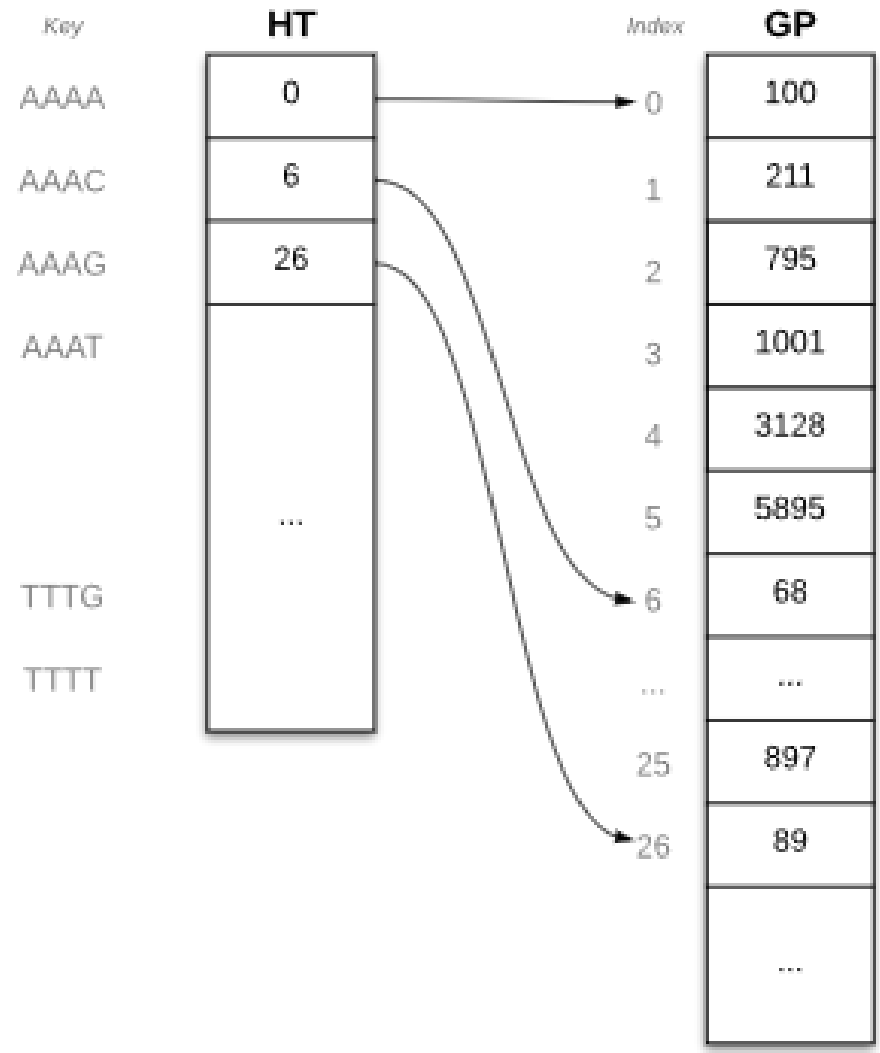
Indexing the reference genome

- k-mer = 13 bp
- Every 3. position
- Only A, C, G, T
- Only + strand

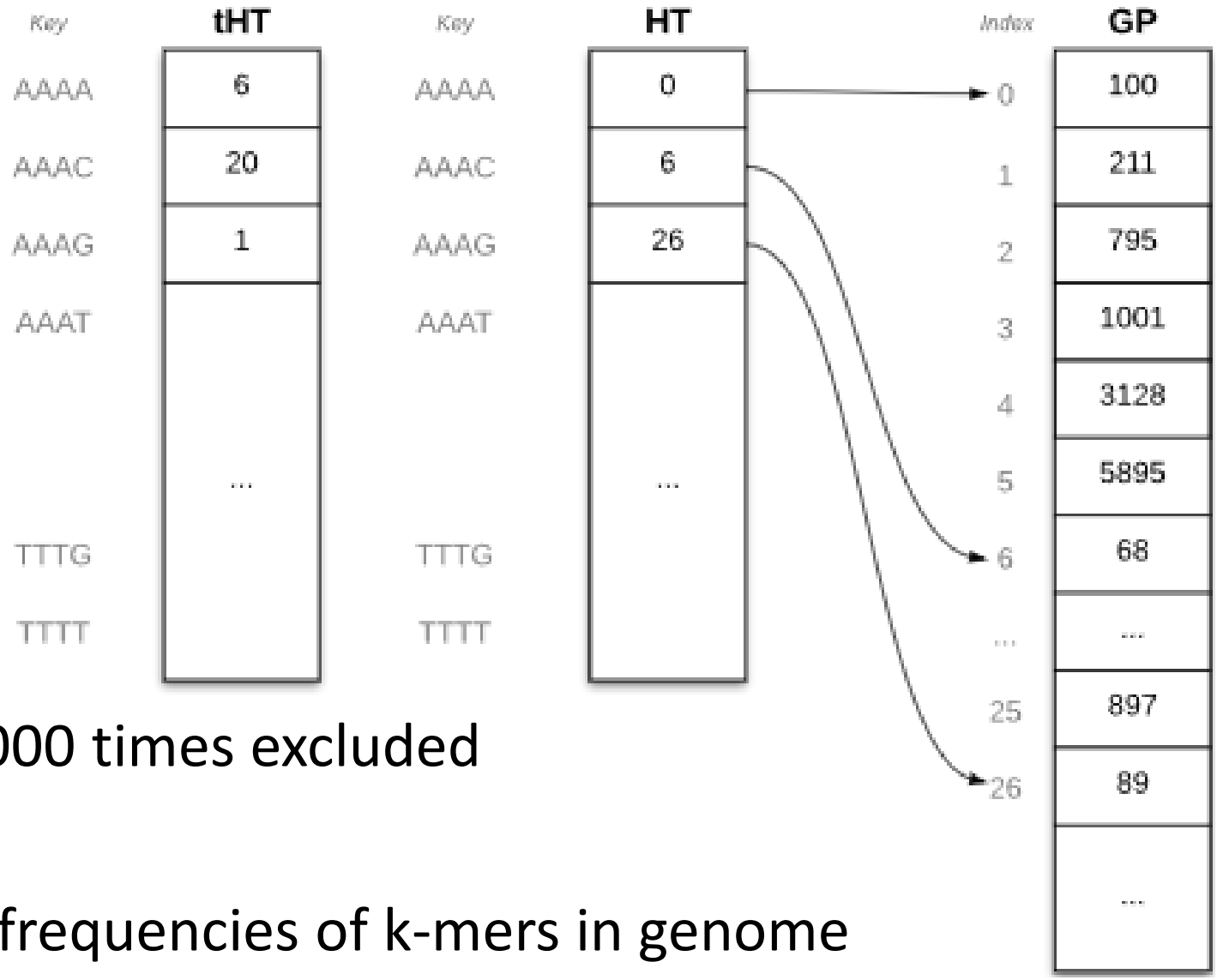


Indexing the reference genome

- GP = genomic positions
  - Consecutively k-mer blocks
  - Saving start position
- HT = Hash table of k-mers
  - Lexicographic ordered
  - Frequency (numbered serially)



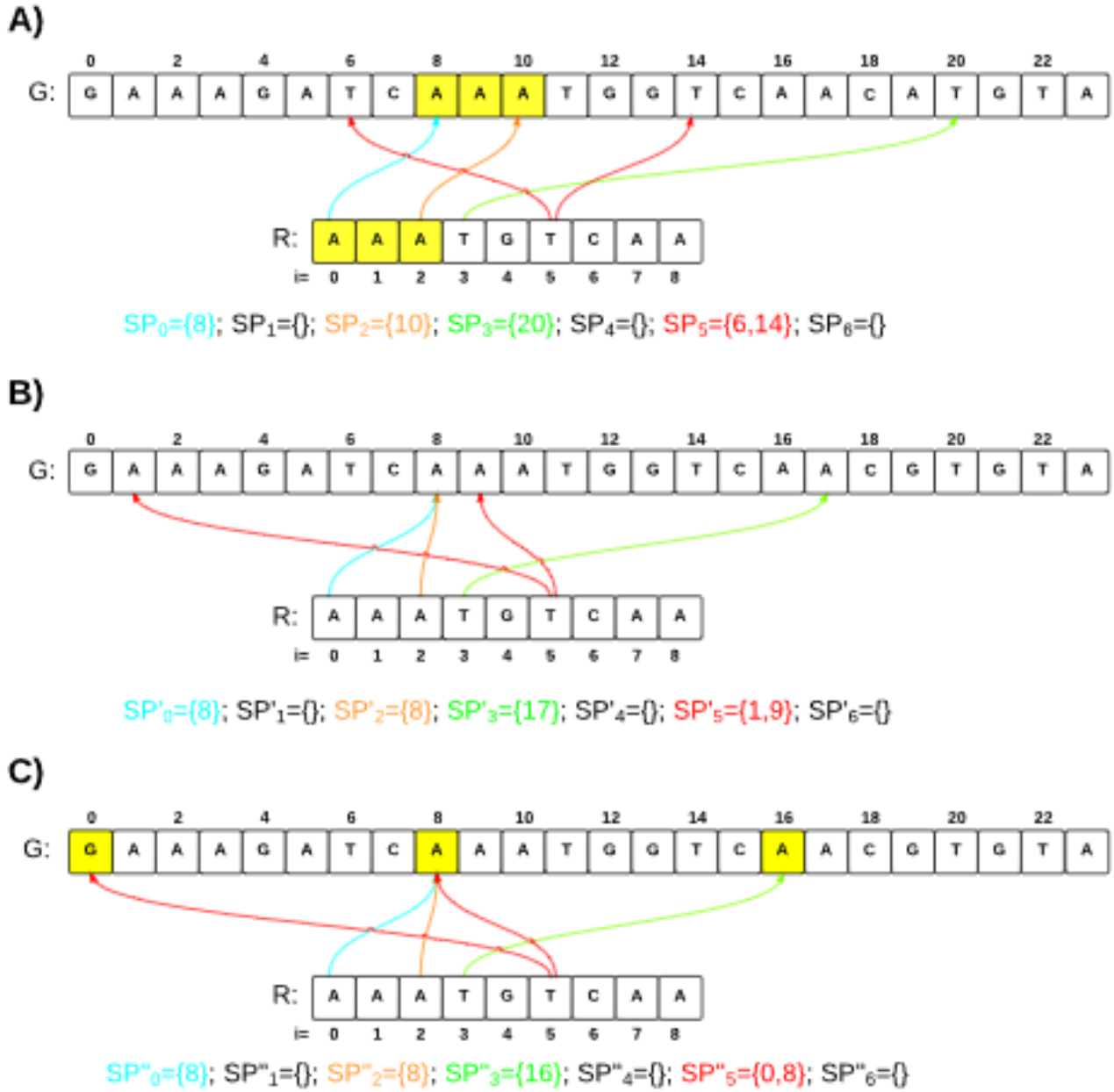
Indexing the reference genome



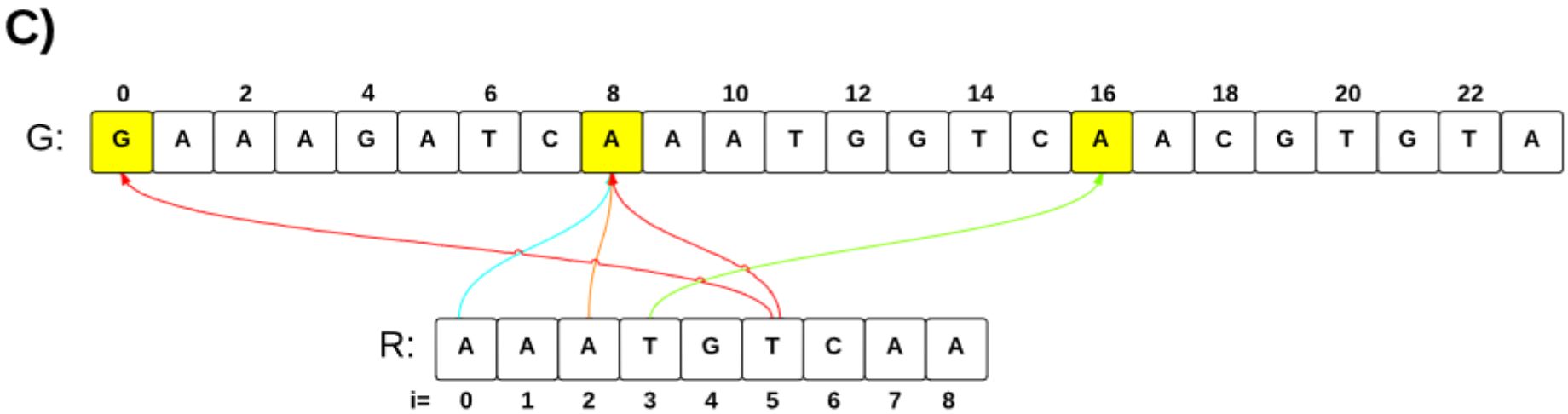
- K-mer >1,000 times excluded
- tHT count frequencies of k-mers in genome

Identification of CMR

- R = read of length 9bp
- G = reference Genome
- Step-size = 2
- k-mer size = 3bp



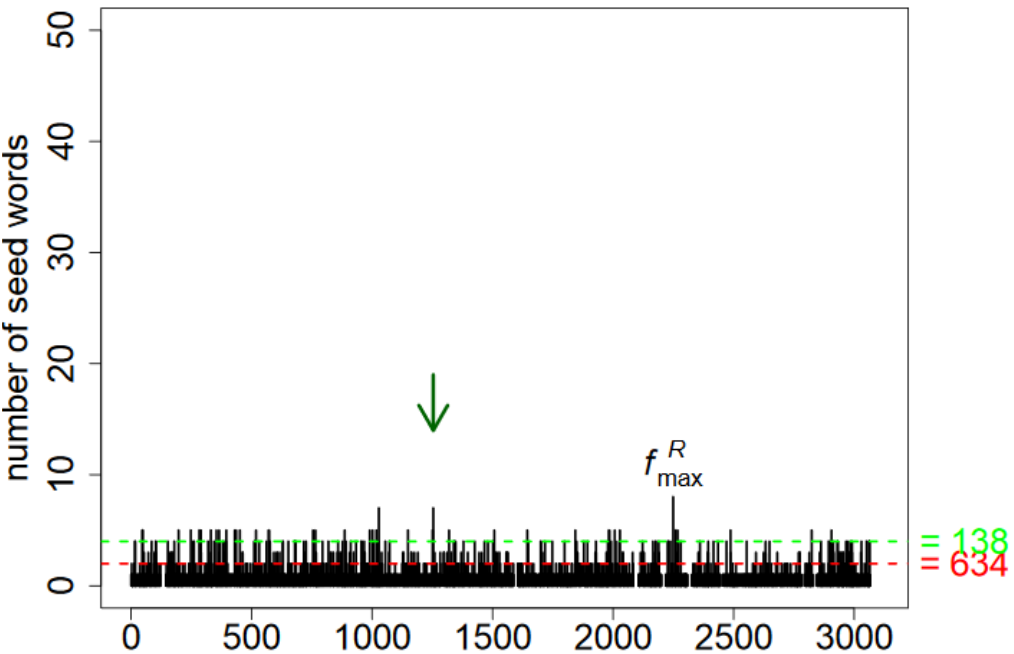
Identification of CMR



$SP''_0=\{8\}$; $SP''_1=\{\}$; $SP''_2=\{8\}$; $SP''_3=\{16\}$; $SP''_4=\{\}$; $SP''_5=\{0,8\}$; $SP''_6=\{\}$

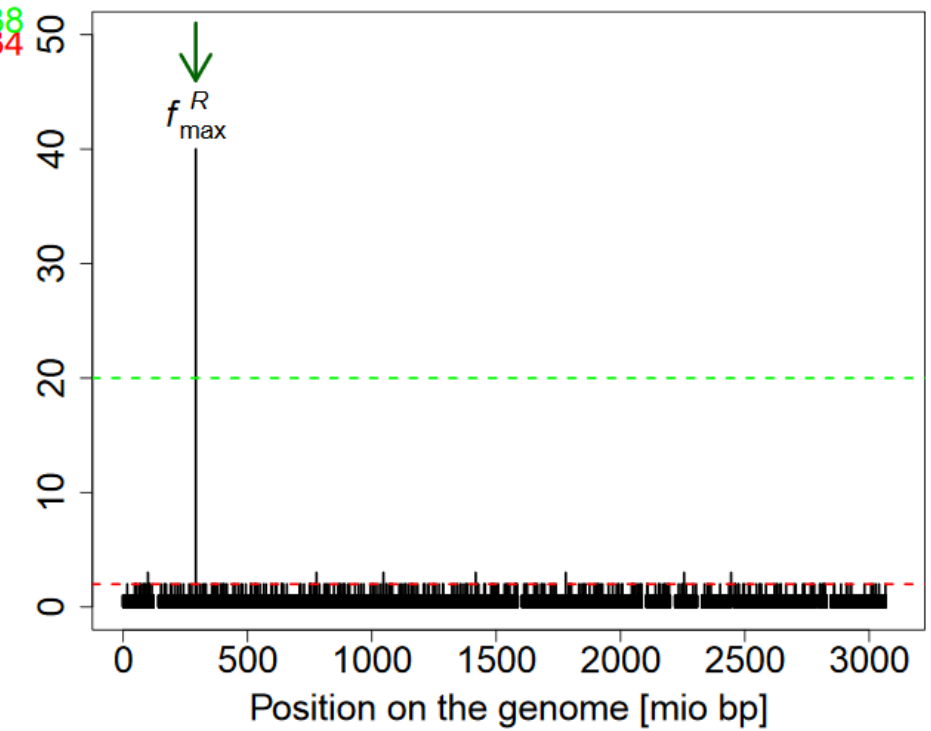
- Position 1 not found due to step-size =2
- Saving startpoints in reference genome
- Shift of readstart to startposition
- Allow modulo 8 (bit-shift operation) due to polymorphic regions to reduce start positions

Computation of CMRs



uninformative

informative



Computation of CMRs

$$\sigma = \frac{\overline{F_{\max}} = \frac{1}{B}\sum_{j=1}^{B}\max\{F_{R_j}\}}{S_{\max} = \left\lceil \frac{l-k+1}{\Delta} \right\rceil}$$

- Calculating the ratio of
  - Average read seed count
  - Perfectly matching read
- σ = 1 → perfect reads
- σ = 0 → very different reads

- Calculating genomic start points
  - F_R → Frequency distribution
  - Genomic position with seed word count above Θ_R

$$\Theta_R = \sigma \max\{F_R\}$$

Computation of alignment scores

1. Calculating the read alignment score

$$p'' - c/2 \text{ to } p'' + l + c/2.$$

2. c defines consecutive insertions deletions dependent on the read length (l)

$$c = 5 + \bar{l} * 0.15$$

3. MASon (Rescheneder et al. 2012) for alignment

BURROWS-WHEELER TRANSFORM BASED ALGORITHMS

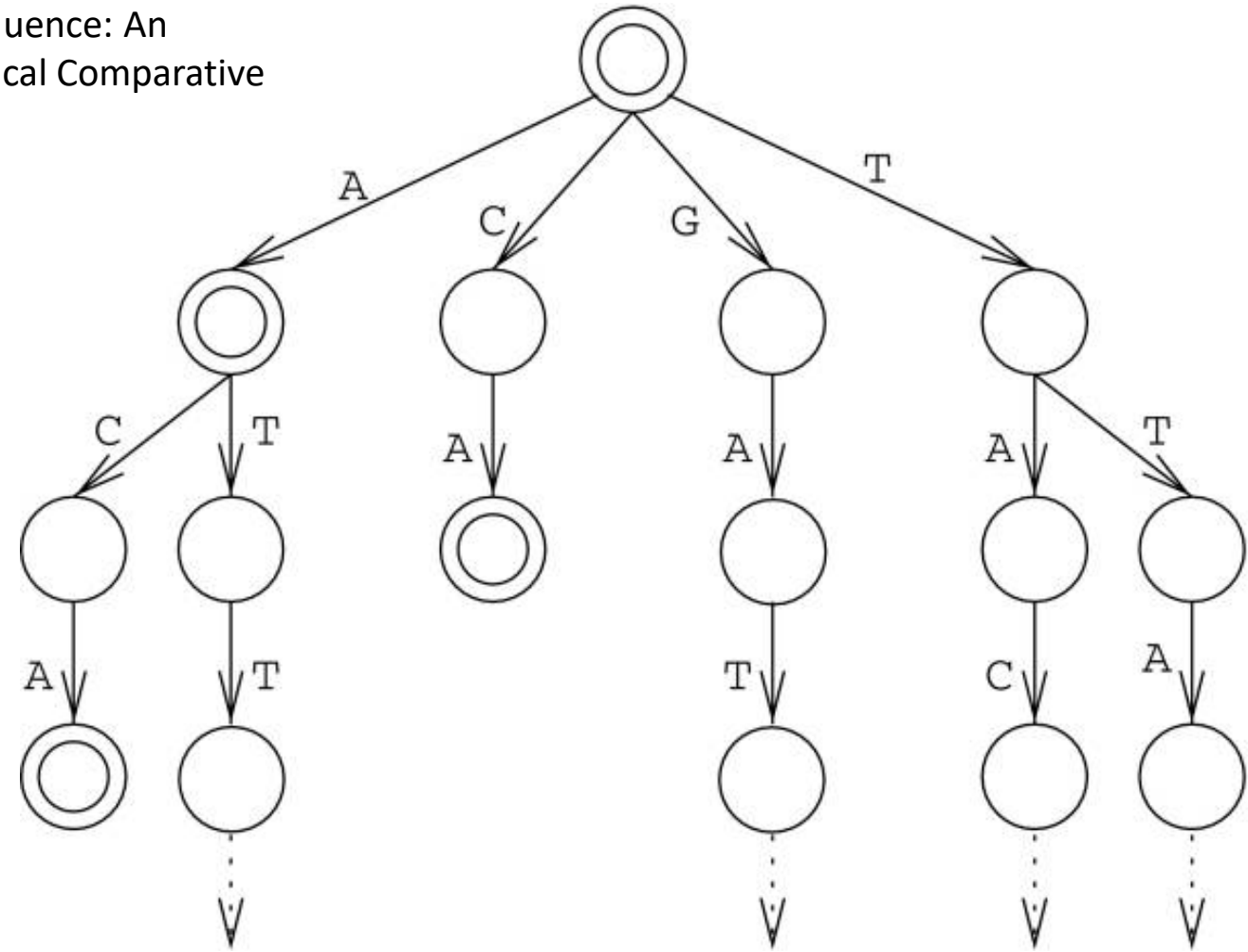
Suffix Trees / Suffix Tries

- **Advantage:**
  - **Hashing performs poor for repeating regions**

- Each suffix of a word is represented as path from leave to root
  - Size of tree proportional to size of genome
  - Building time proportional to size of genome
  - Search time $O(L_r)$

Suffix Trees

Mapping Reads on a Genomic Sequence: An Algorithmic Overview and a Practical Comparative Analysis (2012) Sophie Schbath



- **Advantage:**
  - **Hashing performs poor for repeating regions**
  - **Repeating regions are squeezed into one path**

Suffix Arrays

- Problem of the trees and tries:
  - Large genomes will not fit in the RAM

- Array is the set of suffixes sorted lexicographically
  - Trick 1: save the startposition of the suffix

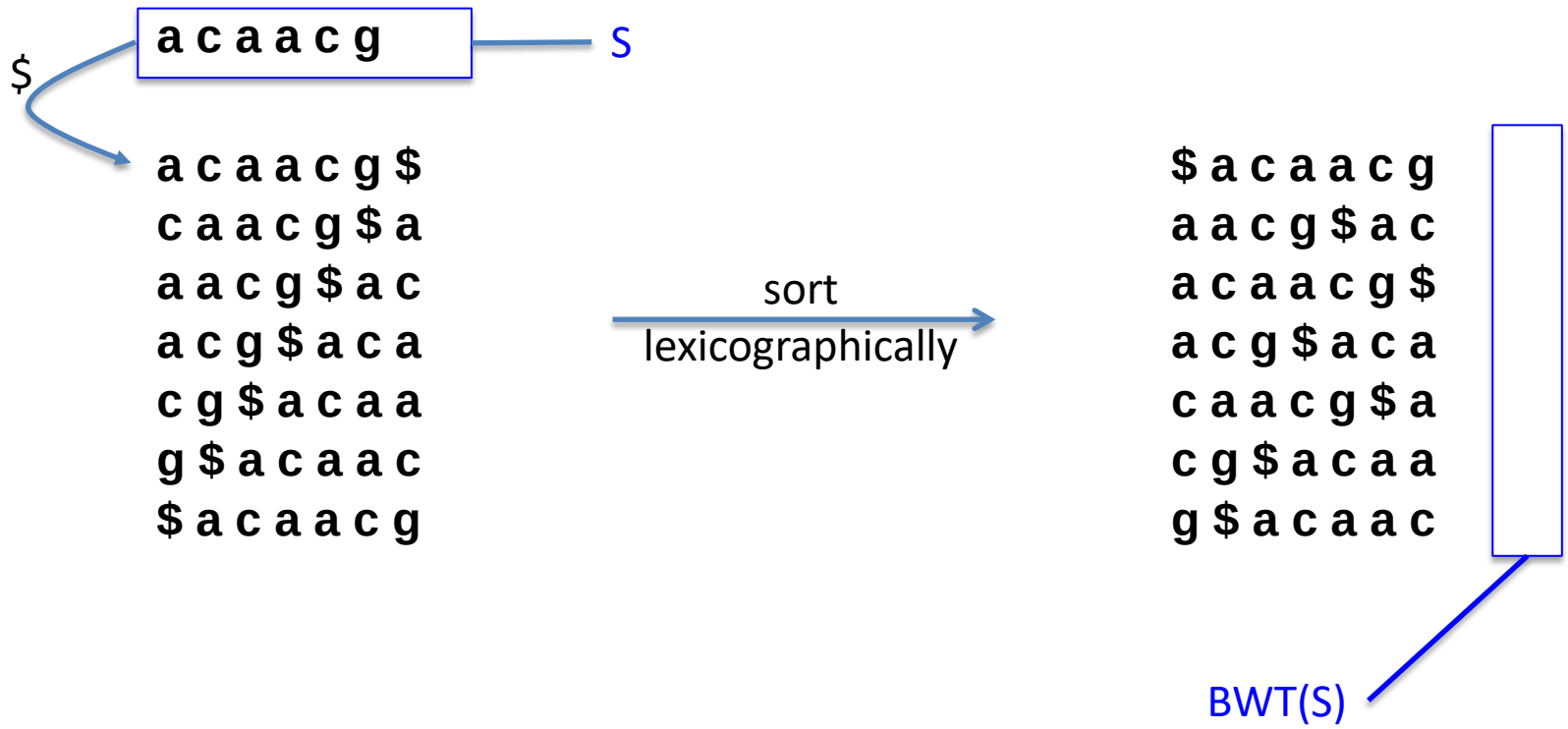
genome	suffixes	sorted suffixes	positions	cylinder suffix array	B–W
GATTACA$	GATTACA$	ACA$	4	ACA$GATT	T
	ATTACA$	ATTACA$	2	ATTACA$G	G
	TTACA$	A$	6	A$GATTAC	C
	TACA$	CA$	5	CA$GATTA	A
	ACA$	GATTACA$	1	GATTACA$	$
	CA$	TACA$	4	TACA$GAT	T
	A$	TTACA$	3	TTACA$GA	A
	$	$	7	$GATTACA	A

Mapping Reads on a Genomic Sequence: An Algorithmic Overview and a Practical Comparative Analysis (2012) Sophie Schbath

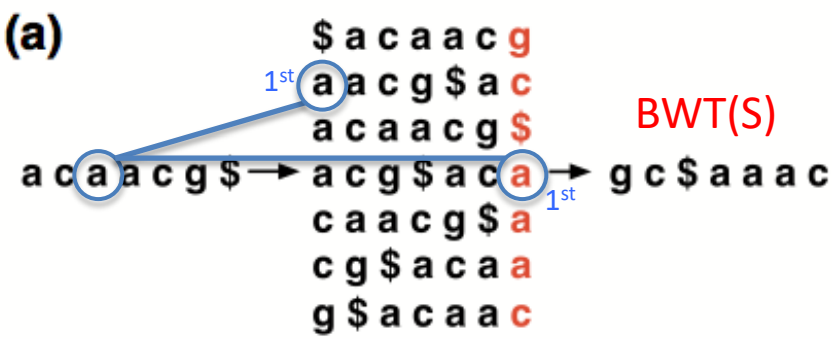
Trick 2: The Burrows-Wheeler Transform

- Invented by David Wheeler in 1983 (bell labs), pub. 1994
- Used in data compression (bzip2)
- Used in 2003 on the human genome to define exact word matches (originally for microarray probe design)
- First used for short read alignment by bowtie, now adopted by bwa (maq author) and SOAP2

The Burrows-Wheeler Transform



Burrows Wheeler Transform (BWT)

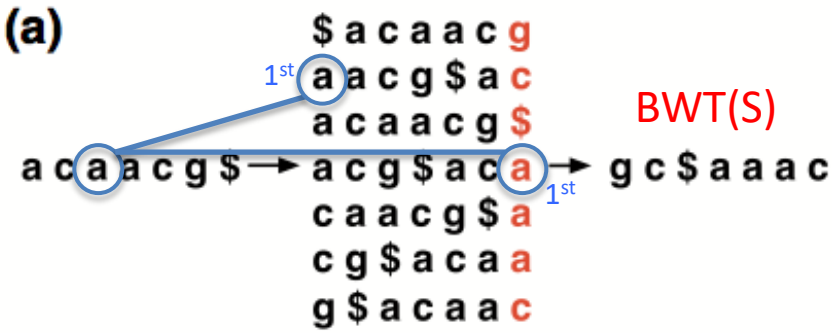


Generate matrix by

1. Appending a *$* to the end of the string *S* that should be indexed. *$* should have 2 properties
   1. it must not occur in the string
   2. it should be lexicographically smaller than any character in S
2. generate all cyclic permutations of S
3. sort the resulting matrix lexicographically (the line beginning with the $ is the first to occur in the matrix.

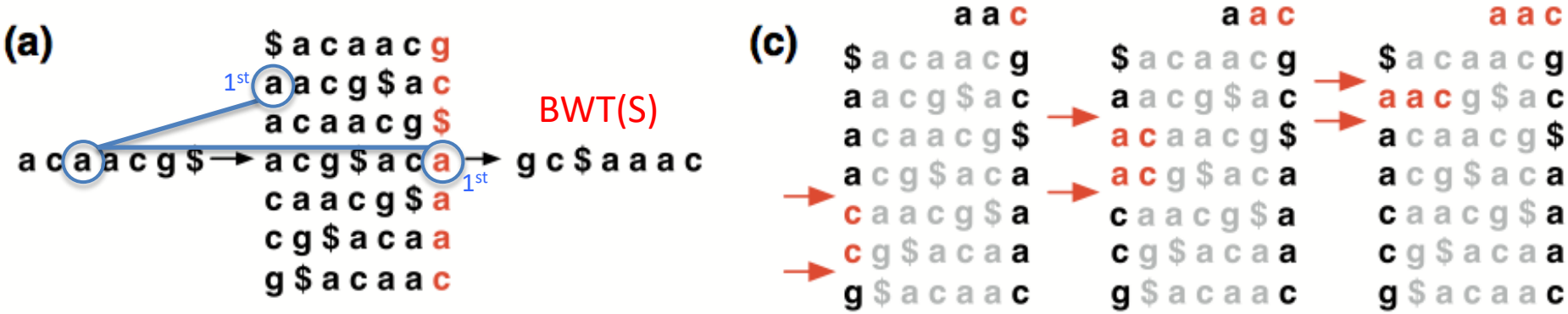
The matrix has the property of last first (LF) mapping: The ith occurrence of character X in the last column corresponds to the same text character as the ith occurrence of X in the first column

Burrows Wheeler Transform (BWT)



The matrix has the property of last first (LF) mapping: This can be used to reconstruct the original text from BWT(S) using the UNPERMUTE algorithm.

Burrows Wheeler Transform (BWT)



The matrix has the property of last first (LF) mapping: This can be used to search for a text within BWT(S) using the EXACTMATCH algorithm.

Key aspects:

1) Matrix is sorted lexicographically. Thus, rows beginning with a given sequence appear **consecutively**.
2) EXACTMATCH algorithm calculates the range of matrix rows beginning with **successively longer suffixes** of the query.
3) At each step, the size of the **range either shrinks or remains the same**.
4) When the algorithm completes, rows beginning with S_0 (the entire query) correspond to **exact occurrences of the query** in the text.

Original string: `ctgaaactggt`

Put a $ on the end

create cyclic rotations of the string...

c	t	g	a	a	a	c	t	g	g	t	$
t	g	a	a	a	c	t	g	g	t	$	c
g	a	a	a	c	t	g	g	t	$	c	t
a	a	a	c	t	g	g	t	$	c	t	g
a	a	c	t	g	g	t	$	c	t	g	a
a	c	t	g	g	t	$	c	t	g	a	a
c	t	g	g	t	$	c	t	g	a	a	a
t	g	g	t	$	c	t	g	a	a	a	c
g	g	t	$	c	t	g	a	a	a	c	t
g	t	$	c	t	g	a	a	a	c	t	g
t	$	c	t	g	a	a	a	c	t	g	g
$	c	t	g	a	a	a	c	t	g	g	t

Alphabetically sort the permuted strings, first column is the "genome dictionary" last column is the Burrows-wheeler transformation

$	c	t	g	a	a	a	c	t	g	g	**t**	11
a	a	a	c	t	g	g	t	$	c	t	**g**	3
a	a	c	t	g	g	t	$	c	t	g	**a**	4
a	c	t	g	g	t	$	c	t	g	a	**a**	5
c	t	g	a	a	a	c	t	g	g	t	**$**	0
c	t	g	g	t	$	c	t	g	a	a	**a**	6
g	a	a	a	c	t	g	g	t	$	c	**t**	2
g	g	t	$	c	t	g	a	a	a	c	**t**	8
g	t	$	c	t	g	a	a	a	c	t	**g**	9
t	$	c	t	g	a	a	a	c	t	g	**g**	10
t	g	a	a	a	c	t	g	g	t	$	**c**	1
t	g	g	t	$	c	t	g	a	a	a	**c**	7

Look up ctgg: start at the end with g, lookup in genome dictionary

top(g) = 6; bot(g) = 8

0	$	c	t	g	a	a	a	c	t	g	g	t
1	a	a	a	c	t	g	g	t	$	c	t	g
2	a	a	c	t	g	g	t	$	c	t	g	a
3	a	c	t	g	g	t	$	c	t	g	a	a
4	c	t	g	a	a	a	c	t	g	g	t	$
5	c	t	g	g	t	$	c	t	g	a	a	a
6	g	a	a	a	c	t	g	g	t	$	c	t
7	g	g	t	$	c	t	g	a	a	a	c	t
8	g	t	$	c	t	g	a	a	a	c	t	g
9	t	$	c	t	g	a	a	a	c	t	g	g
10	t	g	a	a	a	c	t	g	g	t	$	c
11	t	g	g	t	$	c	t	g	a	a	a	c

Does gg exist, and what are top(gg) and bot(gg)?

Yes, gg exists.

top(gg) = top(g) + #g before g-block in bwt = 6 + 1 = 7

bot(gg) = top(gg) + # of gg in genome – 1 = 7 + 1 – 1 = 7

0	$	c	t	g	a	a	a	c	t	g	g	t
1	a	a	a	c	t	g	g	t	$	c	t	g
2	a	a	c	t	g	g	t	$	c	t	g	a
3	a	c	t	g	g	t	$	c	t	g	a	a
4	c	t	g	a	a	a	c	t	g	g	t	$
5	c	t	g	g	t	$	c	t	g	a	a	a
6	g	a	a	a	c	t	g	g	t	$	c	t
7	g	g	t	$	c	t	g	a	a	a	c	t
8	g	t	$	c	t	g	a	a	a	c	t	g
9	t	$	c	t	g	a	a	a	c	t	g	g
10	t	g	a	a	a	c	t	g	g	t	$	c
11	t	g	g	t	$	c	t	g	a	a	a	c

Does tgg exist, and what are top(tgg) and bot(tgg)?

Yes, tgg exists.

top(tgg) = top(t) + #t before gg-block in bwt = 9 + 2 = 11

bot(tgg) = top(tgg) + # of tgg in genome – 1 = 11 + 1 – 1 = 11

0	$	c	t	g	a	a	a	c	t	g	g	t
1	a	a	a	c	t	g	g	t	$	c	t	g
2	a	a	c	t	g	g	t	$	c	t	g	a
3	a	c	t	g	g	t	$	c	t	g	a	a
4	c	t	g	a	a	a	c	t	g	g	t	$
5	c	t	g	g	t	$	c	t	g	a	a	a
6	g	a	a	a	c	t	g	g	t	$	c	t
7	g	g	t	$	c	t	g	a	a	a	c	t
8	g	t	$	c	t	g	a	a	a	c	t	g
9	t	$	c	t	g	a	a	a	c	t	g	g
10	t	g	a	a	a	c	t	g	g	t	$	c
11	t	g	g	t	$	c	t	g	a	a	a	c

Does ctgg exist, and what are top(ctgg) and bot(ctgg)?

Yes, ctgg exists.

top(ctgg) = top(c) + #c before tgg-block in bwt = 4 + 1 = 5

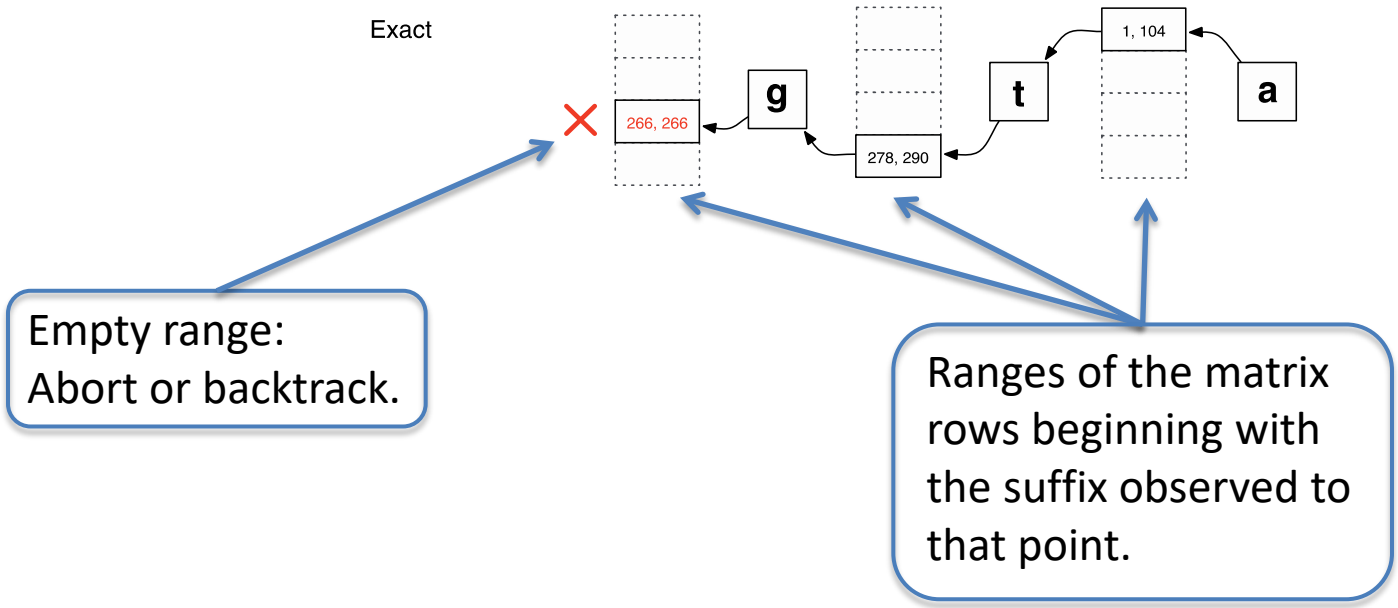
bot(ctgg) = top(ctgg) + # of ctgg in genome – 1 = 5 + 1 – 1 = 5

0	$	c	t	g	a	a	a	c	t	g	g	t
1	a	a	a	c	t	g	g	t	$	c	t	g
2	a	a	c	t	g	g	t	$	c	t	g	a
3	a	c	t	g	g	t	$	c	t	g	a	a
4	c	t	g	a	a	a	c	t	g	g	t	$
5	c	t	g	g	t	$	c	t	g	a	a	a
6	g	a	a	a	c	t	g	g	t	$	c	t
7	g	g	t	$	c	t	g	a	a	a	c	t
8	g	t	$	c	t	g	a	a	a	c	t	g
9	t	$	c	t	g	a	a	a	c	t	g	g
10	t	g	a	a	a	c	t	g	g	t	$	c
11	t	g	g	t	$	c	t	g	a	a	a	c

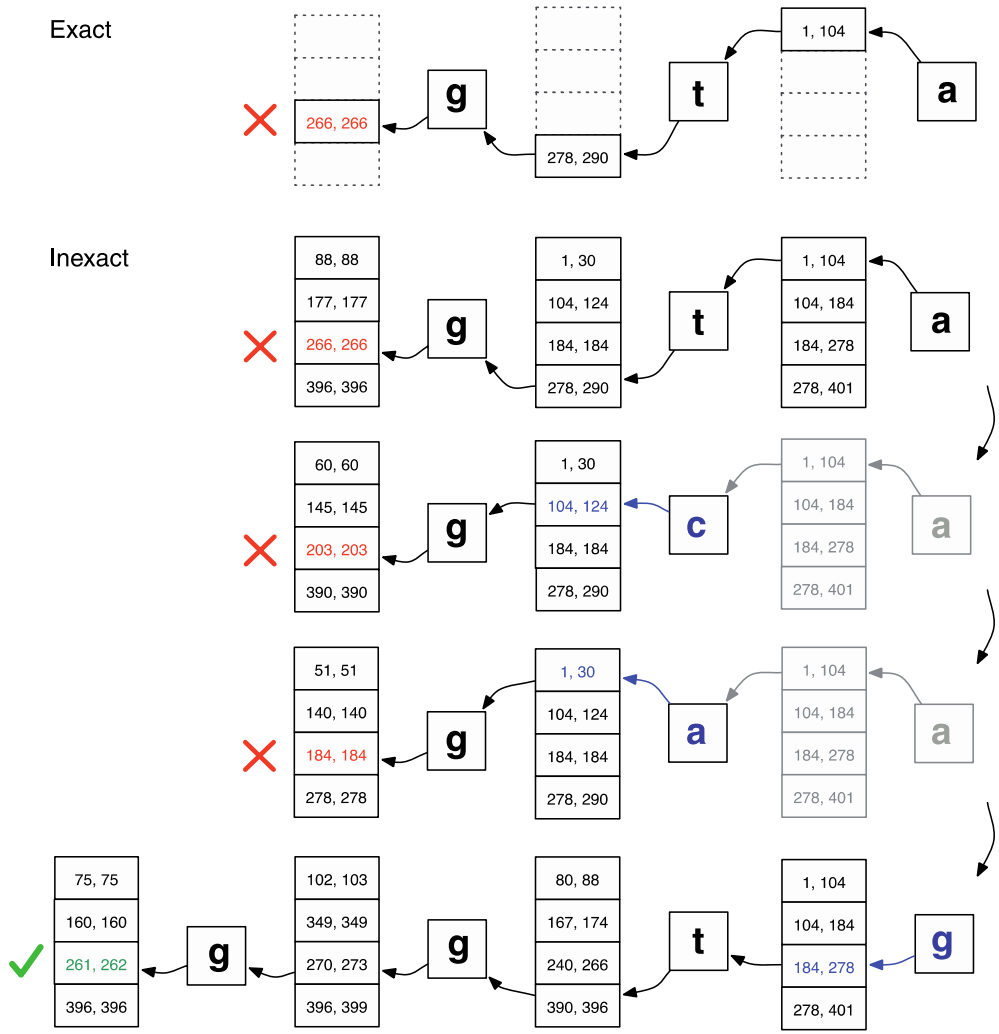
Found ctgg at position 5, which is position 6 in original string ctgaaa**ctgg**t

0	$	c	t	g	a	a	a	c	t	g	g	t	11
1	a	a	a	c	t	g	g	t	$	c	t	g	3
2	a	a	c	t	g	g	t	$	c	t	g	a	4
3	a	c	t	g	g	t	$	c	t	g	a	a	5
4	c	t	g	a	a	a	c	t	g	g	t	$	0
5	c	t	g	g	t	$	c	t	g	a	a	a	6
6	g	a	a	a	c	t	g	g	t	$	c	t	2
7	g	g	t	$	c	t	g	a	a	a	c	t	8
8	g	t	$	c	t	g	a	a	a	c	t	g	9
9	t	$	c	t	g	a	a	a	c	t	g	g	10
10	t	g	a	a	a	c	t	g	g	t	$	c	1
11	t	g	g	t	$	c	t	g	a	a	a	c	7

Coping with mismatches: Query GGTA



Quality-aware, greedy, randomized, depth-first search through the space of possible alignments.



1. The search proceeds similarly to EXACTMATCH
2. If range becomes empty (suffix does not occur in text), the algorithm backtracks and selects an already-matched query position and substitutes a different base there. The EXACTMATCH algorithm resumes from this modified position.
3. The algorithm allows only substitutions that yield a modified suffix that occurs at least once in the text. If there are multiple candidate substitution positions, then the algorithm greedily selects a position with a minimal quality value.
4. Because search is greedy, the first valid alignment is not necessarily the best (Number of mismatches and quality). Bowtie has parameters to cope with this (--best or –all (all alignments).
5. Excessive Backtracking should be avoided. Note, we start from the low quality end...

Coping with mismatches

- Our alignment policy allows a limited number of mismatches and prefers alignments where the sum of the quality values at all mismatched positions is low.
- The search proceeds similarly to EXACTMATCH, calculating matrix ranges for successively longer query suffixes.
- If the range becomes empty (a suffix does not occur in the text), then the algorithm may select an already-matched query position and substitute a different base there. The EXACTMATCH algorithm resumes from this position.
- The algorithm selects only those substitutions that yield a modified suffix that occurs at least once in the text. If there are multiple candidate substitution positions, then the algorithm greedily selects a position with a minimal quality value.

Bowtie: Avoidance of excessive backtracking (assuming 1 mismatch)

Problem: The aligner spends most of its effort fruitlessly backtracking to positions close to the 3' end of the query (error prone).

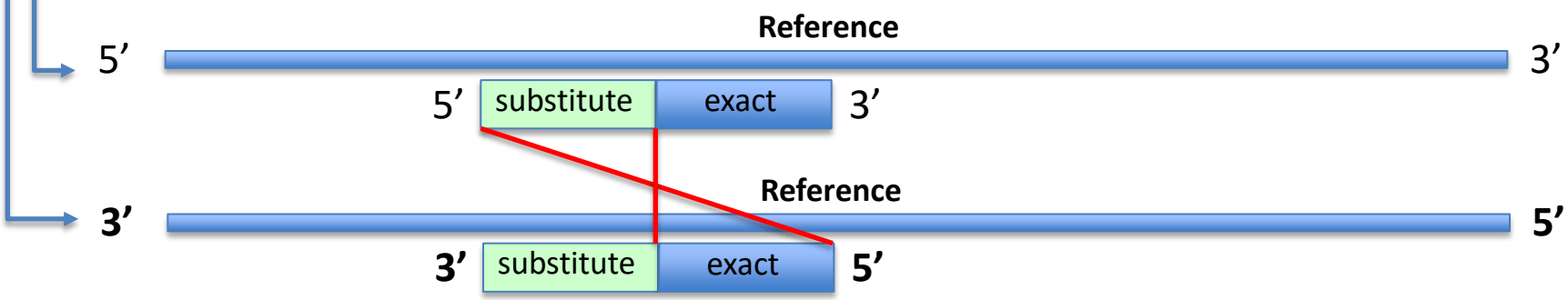
Solution Part1: double indexing (similar to MAQ), using two indices for the genome

- Index 1: BWT of the original genome
- Index 2: BWT of the genome with reversed character order (**not reverse complemented!**)

Solution Part2: The aligner is invoked twice

- First round: Index 1 is used, and the aligner is started with original read with the constraint that it must not substitute a position in the query's right half (3' end).
- Second round: Index 2 is used, and the aligner is started with the reversed read, again with the constraint that it must not substitute a position in the reversed query's right half (originally and still the 5' end).

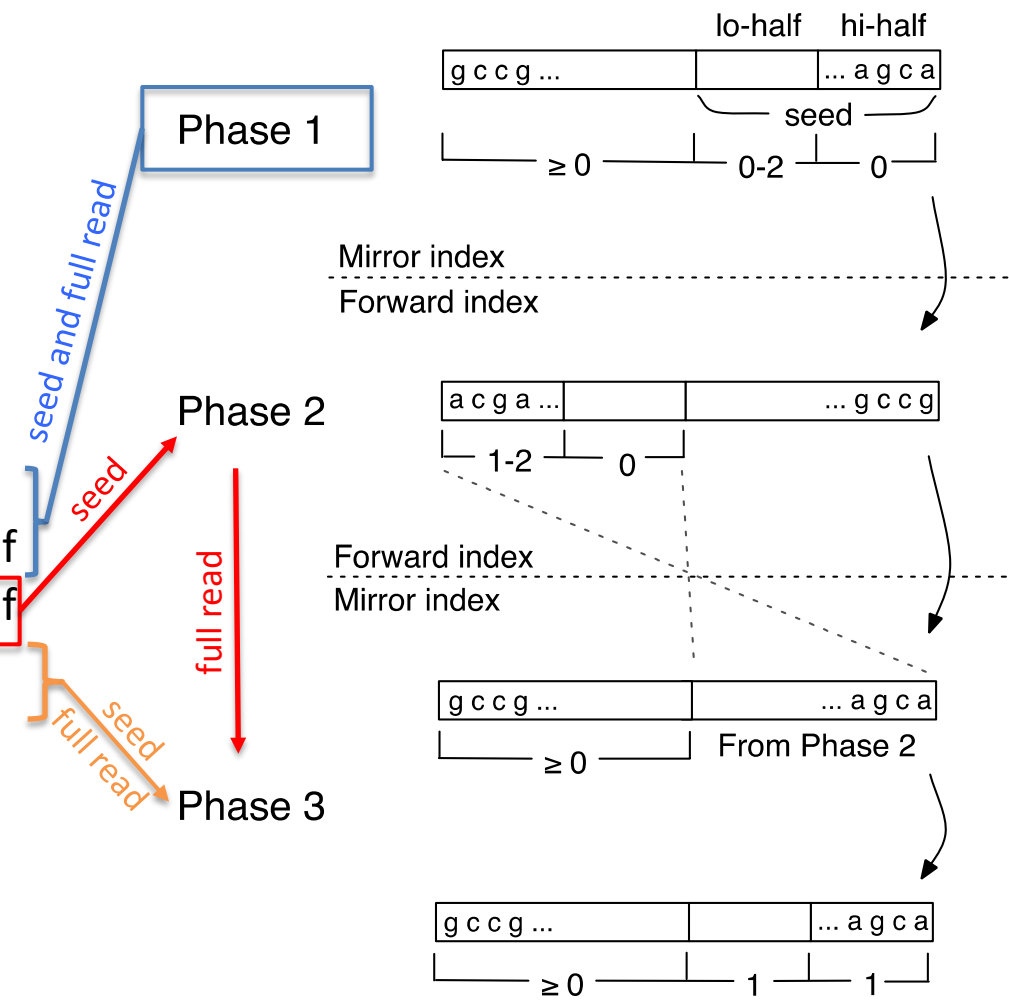
Solution Part3: set a hard upper limit of backtracks to be performed.



The three phases of Bowtie

In the case of 2 (or more mismatches):

- Bowtie uses the first 28 bp as seed
- The seed is split into a high quality 5' half (hi-half) and a low quality 3' half (lo-half)
- For up to 2 mismatches we have four scenarios:
  1. no mismatches in seed
  2. 1-2 mismatches only in the lo-half
  3. 1-2 mismatches only in the hi-half
  4. 1 mismatch each in hi- and low-half
- Any number of mismatches can occur in non-seed part (subject to other thresholds).



Changes in Bowtie2

Supports gapped, local, and paired-end alignment modes:

- For reads longer than about 50 bp Bowtie 2 is generally faster, more sensitive, and uses less memory.

- Bowtie 2 supports local alignment, which doesn't require reads to align end-to-end. Local alignments might be "trimmed" ("soft clipped") at one or both extremes in a way that optimizes alignment score.

- There is no upper limit on read length in Bowtie 2.

- Bowtie 2 allows alignments to overlap ambiguous characters (e.g. Ns) in the reference.

Bowtie2: Seed extraction & alignment

Seed extraction:

- Substrings of the read ("seed strings") are extracted at regular intervals along the read and its reverse complement.
- Seed strings are contiguous (i.e. they are not spaced seeds) and may or may not overlap each other.

FM Index assisted seed alignment :

- Find ungapped alignments for each based on Bowtie1
- Seed strings can be aligned with up to 1 mismatch.

Bowtie2: Seed alignment priorization & alignment

Priorization:

- "seed-hit range" → A seed-hit range describes a range of rows in the Burrows-Wheeler matrix that begin with a reference substring that is within 0 or 1 mismatches of the seed substring.
- Bowtie 2 proceeds by repeatedly selecting a row in a random weighted fashion using these weights.

Alignment:

- Bowtie 2 extracts flanking characters from the reference
- Solves a rectangular dynamic programming problem to find high-scoring full alignments in the vicinity of the seed hit.

Watch out for the following…

- Up to 2 mismatches in first 28 bases reported by Maq (default) and 1-2 mismatches by Bowtie1/2 in the seed.
- If >=1 matching seed regions exist for one read Bowtie2 has a dynamic programming effort limit of 15. After 15 attempts not performing better than the best so far.
- Bowtie outperforms Bowtie2 in case of short reads (<=50 nt) in some cases.
- Bowtie convert N to A, C, G or T randomly / Bowtie2 accepts N's


THANK
YOU
GRACIAS
ARIGATO
SHUKURIA
DANKSCHEEN
JUSPAXAR
GOZAIMASHITA
EFCHARISTO
KOMAPSUMNIDA
MALAKE
GRAZIE
MEHRBANI
PALDIES
SUKSAMA
EKHMET
TASHAKKUR ATU
YAQHANYELAY
TINGKI
BÏYAN
SHUKRIA
BOLZÏN
MERCI