COMP90016 – Computational Genomics
Sequence Alignment

Department of Computing and Information Systems

The University of Melbourne

Motivation

- The lectures have introduced sequencing as the basic technology of genomics
  - Sequencing provides us with digital representations of DNA/RNA/Amino acid chains.
- What to do with these sequences?
  - Compare them to known genes, genomes!

Sequence alignment

"Alignment will be the Google^{TM} of genomics – finding all interesting connections between diverse sequences."

-- S. Batzoglou, *Briefings in Bioinformatics* **6**(1), 6-22, 2005

Alignment

Sequence alignment is the task of (inexact) string matching.
In genomics, this technique is used in various different ways:

Sequence Comparison

- Are these two bits of DNA similar, of a common ancestor?
- Are these two sequences of amino acids (proteins) similar?
- Depending on the extent of evolution since common ancestry, differences can be large -> low level of sequence conservation.
- Where have we seen DNA such as this before? For example, a clinical sample of a bacteria.

Read Alignment

- Where in the reference genome are the most likely origins of the sequenced reads?
- Find the best alignment and deduce the correct position.
- Since reference and sequenced DNA are not identical, read alignment has to do inexact string comparison.

All of these tasks need to compare strings of various origin with each other and determine similarity/difference.

Issues specific to biological sequences

- *Approximate* matching, to accommodate:
  - Sequencing errors.
  - Evolution (within and between species).
- What are we looking for?
  - Similar strings.
  - Partial (domain) matching.
  - How to assess relative similarity?
  - A measure of what is closest.

Measuring the distance between two sequences

- Hamming distance:

  $\sum_{i=1}^{n} m(a_i - b_i)$

  - $m(a_i-b_i)=1$ when a_i != b_i
  - $m(a_i-b_i)=0$ when a_i == b_i

- Edit distance $=_{definition}$ minimum number of edit operations to transform one sequence to another.
  - Edit operations:
    - Insert/delete
    - Substitute
  - Examples:
    - AAAA -> DRMM -> CAA
    - ATAC -> MIIMRM -> ACGTGC

V.I. Levenshtein, Problems in Information Transmission **1**(1), 8-17, 1965. Original in Russian prior to this.
Edit distance is also referred to as Levenshtein Distance

Alignment Example

- Compute the edit distance between these strings (using any method):
    - Clarinet
    - Cabinet
- Show an edit transcript, and an alignment.
  Edit transcripts transform one string into the other by a sequence of
  - M (match)
  - R (replace)
  - D (delete)
  - I (insert)

Alignment Example

- The edit (Levenshtein) distance between the two strings is 2.
- Alignment:

```
S1: C l a r i n e t
    |   |   | | | |
S2: C - a b i n e t
```

- Edit transcript:
  S1 -> (MDMRMMMM) -> S2
- Note that the Hamming distance is not defined, since the strings are not of the same length.

Measuring the distance between two DNA sequences

- Sequencing errors can be insertions or deletions of bases.
  - Depending on the technology these can be rare (1 in a 1000 reads) or frequent (most reads).
- Evolution can change DNA by inserting, deleting, or moving long and short sub-sequences.
  - In genomics, insertions and deletions are referred to as *indels.*
  - Transposable elements.
  - Single-nucleotide indels.
  - Deletions.
  - All of these are frequent in many organisms (including humans)

Edit Distance vs. Biology

- In genomics edit distance is the way to go over the other metrics. Why?
  - Substitutions are a common mutation (discussed in a later lecture called SNPs) – these are identified nicely through Hamming distance etc.
  - But indels are intractable for the "i=1 to n metrics".
- However, edit distance is very costly to calculate compared to Hamming distance.

Sequence alignment with edit distance

- How then does string matching with gaps work, and how do we calculate the respective edit distance?
- How did you find the edit distance between *Clarinet* and *Cabinet*?
  - How do you know the alignment was optimal?
  - How can you generalise your approach to convert it into a computer algorithm?

Implementing Edit Distance in Python

```
def lev(a, b):
    if("" == a):
        return len(b) # returns if a is an empty string
    if("" == b):
        return len(a) # returns if b is an empty string
    return min(lev(a[:-1], b[:-1])+(a[-1] != b[-1]),
               lev(a[:-1], b)+1,
               lev(a, b[:-1])+1)
```

What is the computational complexity of this program?

A Word on Complexity

- Computer Science is interest in the complexity of algorithms (programs).
- Complexity is a theoretical consideration of the runtime and scaling of runtime.
- Consider this program:

```
for i in range(n):
    print i
```

- What is its complexity?
  - O(n)

Algorithmic Complexity

- The O symbol describes a function that is limiting the scaling of the analysed code.
  - The code is not as bad as this function – no matter what the variables.
- Another example:

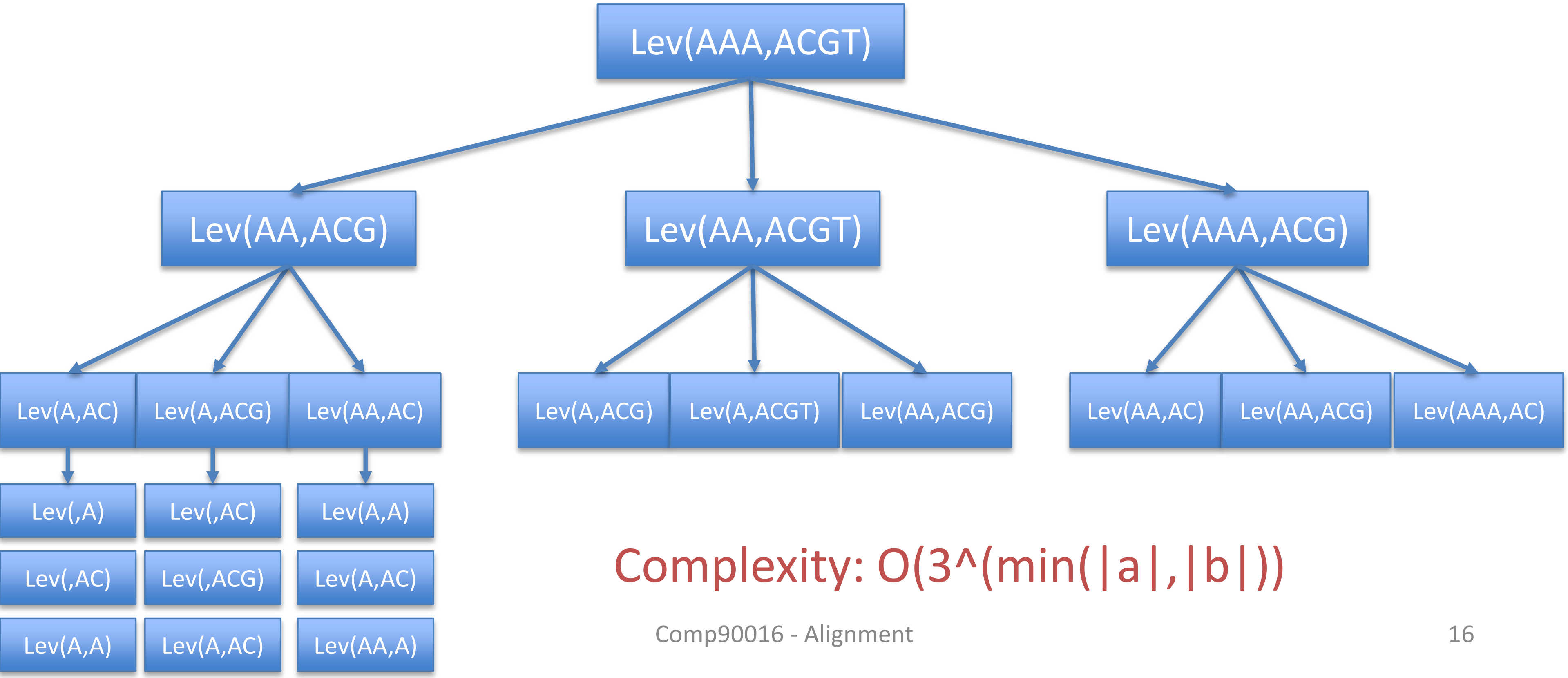
```
for i in range(n):
    for j in range(i):
        print j
```

- The complexity is O(n*n)
  - Even though only the last iteration of the outer for-loop has n operations, increasing n still scales quadratically.

For more Information - Big O notation:
https://en.wikipedia.org/wiki/Big_O_notation

Analysing The Call Tree For *lev*

- Assume Two strings of length 3 and 4 to be compared. For example, a=AAA, b=ACGT



Complexity: O(3^(min(|a|,|b|))

Complexity Issues – a thought example

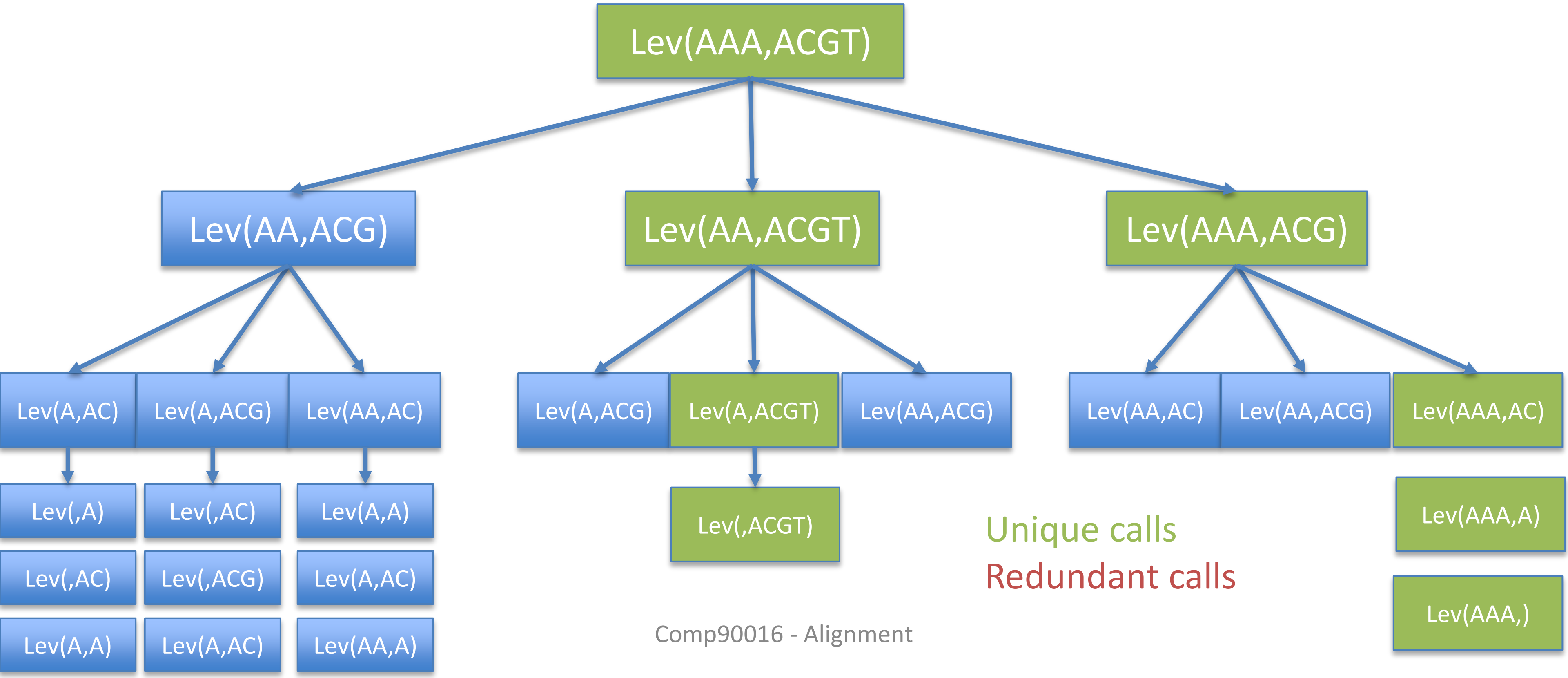
- Let us consider *lev* for the use of aligning a sequencing read to the human reference genome.
- Let's say a computer can compute the minimum of three values in 3 operations.
- On a 3GHz computer with 3 cores, a read of 100bp, and a genome of 3*10^9bp, we arrive at:
- 3ops*3^(min(100bp, 3*10^9bp)/(3cores * 3*10^9 ops/s) = 5*10^30 years for a single read.
- A sequencer produces 100M reads and more.

Complexity and Waste of *lev*

- Question:
  - How many times does the lev algorithm calculate a value that is not redundant?
  - Only the string combinations that have one of the full input strings - |S1|+|S2| + 1
  - All other inputs are redundant at least twice (depending on level – short strings get called more often).

Analysing The Call Tree For *lev*

- Assume Two strings of length 3 and 4 to be compared. For example, a=AAA, b=ACGT



Complexity and Waste of *lev* (2)

- Question:
  - How many times does the lev algorithm calculate a distance that it could already know?
- How many unique lev calls are there for any two strings S1 and S2?
  - There are |S1| prefixes of S1 plus the empty prefix– each can be combined with |S2| + 1 prefix of S2.
  - > There are |S1+1|*|S2+1| distinct calls to lev in the tree.

Towards a More Efficient Algorithm

- For two strings S_1 and S_2,
  - Define D as a distance matrix to remember intermediate results.
  - D is a 2-dimensional matrix for comparing 2 strings, a 3-dimensional matrix for comparing 3 strings, *etc.*
  - Comparing two strings S_1 and S_2, $D(i,j)$ is the edit distance between substrings $S_1[1..\mathbf{i}]$ and $S_2[1..\mathbf{j}]$.
  - $D(n,m)$ is the edit distance between S_1 and S_2.

Distance Matrix Example

	A	C	G	T
A	🟧	🟧	🟧	🟩
A	🟧	🟧	🟥	🟩
A	🟩	🟩	🟩	🟩 Final result

Unique values (single use)

Redundant value (used 3 times)

There are three paths that depend on this edit distance

Redundant values (used >3 times)

Observation:

There are so many redundant calls that we can reduce the number of computations from 3^n to n^2!!

	A	C	G	T
A	Lev(A,A)			
A				
A				

Lev(A,A) = min(**lev(,) + 0**, **lev(A,) + 1**, **lev(,A) + 1**)
min(0+0, 1+1, 1+1) = 0

	A	C	G	T
A	0			
A	Lev(AA,A)			
A				

Lev(AA,A) = min(**lev(A,) + 0**, **lev(AA,) + 1**, **lev(A,A) + 1**)
= min(1+0, 2+1, 0+1) = 1

	A	C	G	T
A	0	1		
A	1	Lev(AA,AC)		
A				3

Lev(AA,AC) = min(**lev(A,A) + 1**, **lev(AA,A) + 1**, **lev(A,AC) + 1**)
= min(0+1, 1+1, 1+1) = 1

Edit distance matrix

- We will work with 2 strings, but the method is more generally applicable.
- *D* is the distance matrix
- for strings S_1 (length *n*) and S_2 (length *m*), where:
  - $D(i,j)$ is the edit distance between $S_1[1..\mathbf{i}]$ and $S_2[1..\mathbf{j}]$. (substrings)
  - $D(n,m)$ is the edit distance between S_1 and S_2 (full strings)

Dynamic programming algorithm (DPA)

- Dynamic programming allows computation of lowest cost alignments in *two stages*:
    - Compute (least) distance for each pair of substrings, using recurrence relationships.
    - Traceback.
- Underlying idea:

  $D(i,j)$ can be computed from
  - $D(i-1,j-1)$, $D(i-1,j)$, $D(j-1,i)$
  - Characters $S_1[i]$ and $S_2[j]$
  - Cost function (implicit or explicit)

Dynamic Programming Motivation

```
Remember :return min(lev(a[:-1], b[:-1])+(a[-1] != b[-1]),
                     lev(a[:-1], b)+1,
                     lev(a, b[:-1])+1)
```

- ACGGATGCTCCGT
- GGATGCTGAGCTCC
- To compute the distance between the prefixes ending in G, we do not need to know the whole history of the comparison.
- We just need to know the previous distances D(i-1,j-1), D(i-1,j), and D(i,j-1).
- For the first row and column, some of these are defined by the "return len(b)" (or a) term.

DPA recurrence relation to calculate edit distance: (weight of all operations = 1)

- $D(i,j) = \min \begin{cases} D(i-1,j) + 1 & \textit{(deletion)} \\ D(i,j-1) + 1 & \textit{(insertion)} \\ D(i-1,j-1) + t(i,j) & \textit{(mis/match)} \end{cases}$

  - $t(i,j)=1$ where $S_1[i]!=S_2[j]$ (mismatch, costs 1)
  - $t(i,j)=0$ where $S_1[i]=S_2[j]$ (match, no cost)

- Initialization:
  - $D(i,0) = i$ (cost for leading deletions)
  - $D(0,j) = j$ (cost for leading insertions)
- Distance(S1,S2) = D(n,m)

DPA Examples

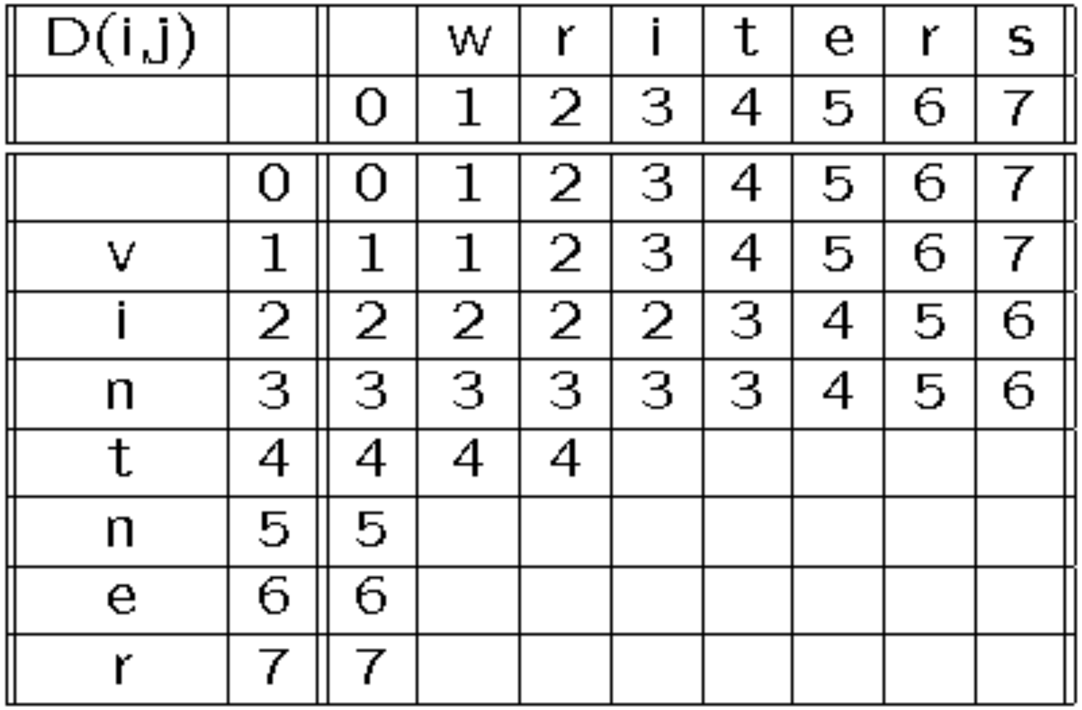
- Example 1:
  - S1: abc
  - S2: aec
- Example 2:
  - S1: bdxy
  - S2: cxy
- Example 3:
  - S1: vintner
  - S2: writers

DPA example

Example: S_1: vintner, S_2: writers

D(i,j)			w	r	i	t	e	r	s
		0	1	2	3	4	5	6	7
	0	0	1	2	3	4	5	6	7
v	1	1							
i	2	2							
n	3	3							
t	4	4							
n	5	5							
e	6	6							
r	7	7							

DPA Example



D(i,j)			w	r	i	t	e	r	s
		0	1	2	3	4	5	6	7
	0	0	1	2	3	4	5	6	7
v	1	1	1	2	3	4	5	6	7
i	2	2	2	2	2	3	4	5	6
n	3	3	3	3	3	3	4	5	6
t	4	4	4	4					
n	5	5							
e	6	6							
r	7	7							

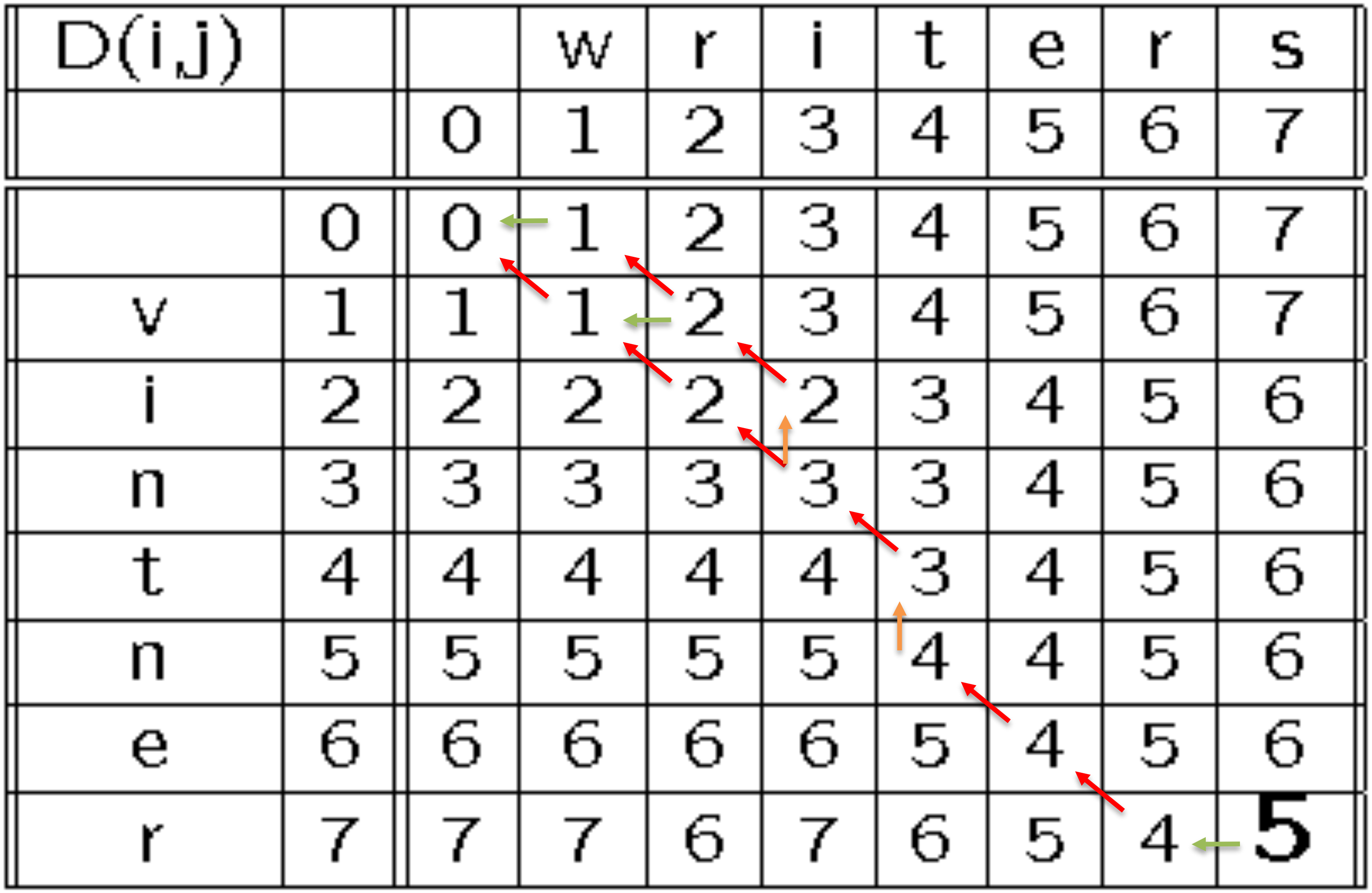
DPA Example

D(i,j)			w	r	i	t	e	r	s
		0	1	2	3	4	5	6	7
	0	0	1	2	3	4	5	6	7
v	1	1	1	2	3	4	5	6	7
i	2	2	2	2	2	3	4	5	6
n	3	3	3	3	3	3	4	5	6
t	4	4	4	4	4	3	4	5	6
n	5	5	5	5	5	4	4	5	6
e	6	6	6	6	6	5	4	5	6
r	7	7	7	6	7	6	5	4	**5**

DPA: Finding alignment

- While computing scores, leave pointer to cell(s) that showed min cost (at an increased memory cost).
- Traceback from cell *D(n,m)* to *D(0,0),* following pointers.
- *n.b.*
  - Sometimes there will be >1 pointers
  - *n.b.* Sometimes there will be >1 least cost alignments
- Or, from cell D(n,m) calculate which neighbouring cell(s) produced this value, until reaching D(0,0)

Traceback Example (without pre-computed pointers)



D(i,j)			w	r	i	t	e	r	s
		0	1	2	3	4	5	6	7
	0	0	1	2	3	4	5	6	7
v	1	1	1	2	3	4	5	6	7
i	2	2	2	2	2	3	4	5	6
n	3	3	3	3	3	3	4	5	6
t	4	4	4	4	4	3	4	5	6
n	5	5	5	5	5	4	4	5	6
e	6	6	6	6	6	5	4	5	6
r	7	7	7	6	7	6	5	4	**5**

- Alignments

```
writ-ers     RRRMIMMD
   | ||
vintner-

wri-t-ers    DRMIMIMMD
  | | ||
-vintner-

wri-t-ers    RDMIMIMMD
  | | ||
v-intner-
```

Arbitrary cost DPA Needleman-Wunsch algorithm

- $D(i,j) = \min \begin{cases} \bullet D(i-1,j) + c_i & \textit{(insertion)} \\ \bullet D(i,j-1) + c_d & \textit{(deletion)} \\ \bullet D(i-1,j-1) + t(i,j) & \textit{(mis/match)} \end{cases}$

•$t(i,j)=c_r$ where $S_1[i]!=S_2[j]$

•$t(i,j)=c_m$ where $S1[i]=S2[j]$

•Initialization:

•$D(i,0) = i{*}c_d$

•$D(0,j) = j{*}c_i$

•$\text{Distance}(S_1,S_2) = D(n,m)$

S.B.Needleman and C.D.Wunsch, J.Mol.Biol. 48, 443-453, 1970.

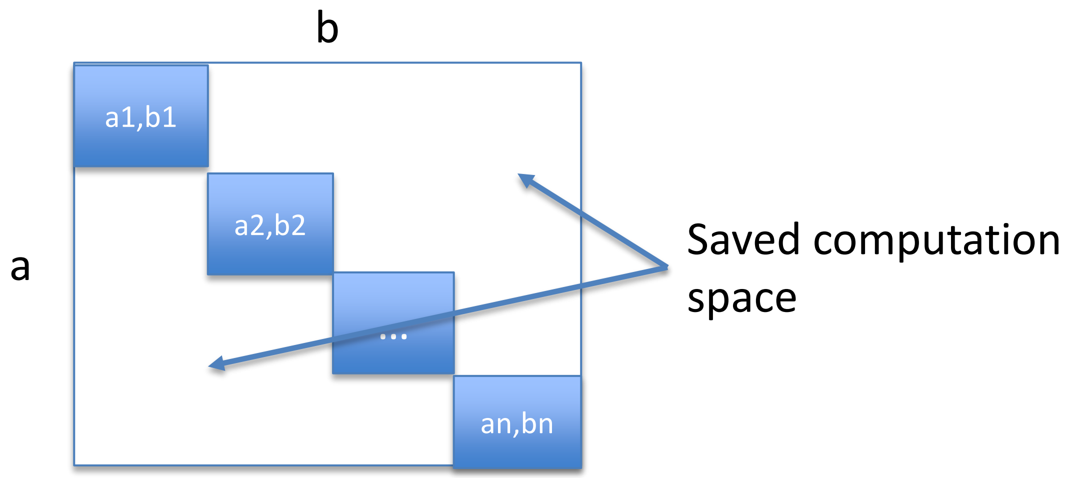
Needleman-Wunsch complexity analysis

- For 2 strings or length *n* and *m:*
  - Time complexity:
    - Fill in table: O(nm)
    - Backtrace (one best alignment): O(n+m)
    - Overall: O(nm)
    - What about backtrace all best alignments?
  - Space complexity:

    O(nm)
- With our complexity example from before (100bp read, human genome…), and assuming 3 operatons to calcualte the minimum, we get:
  - 3*100*3*10^9/(3*3*10^9) = 100s (317 years for 100M reads)

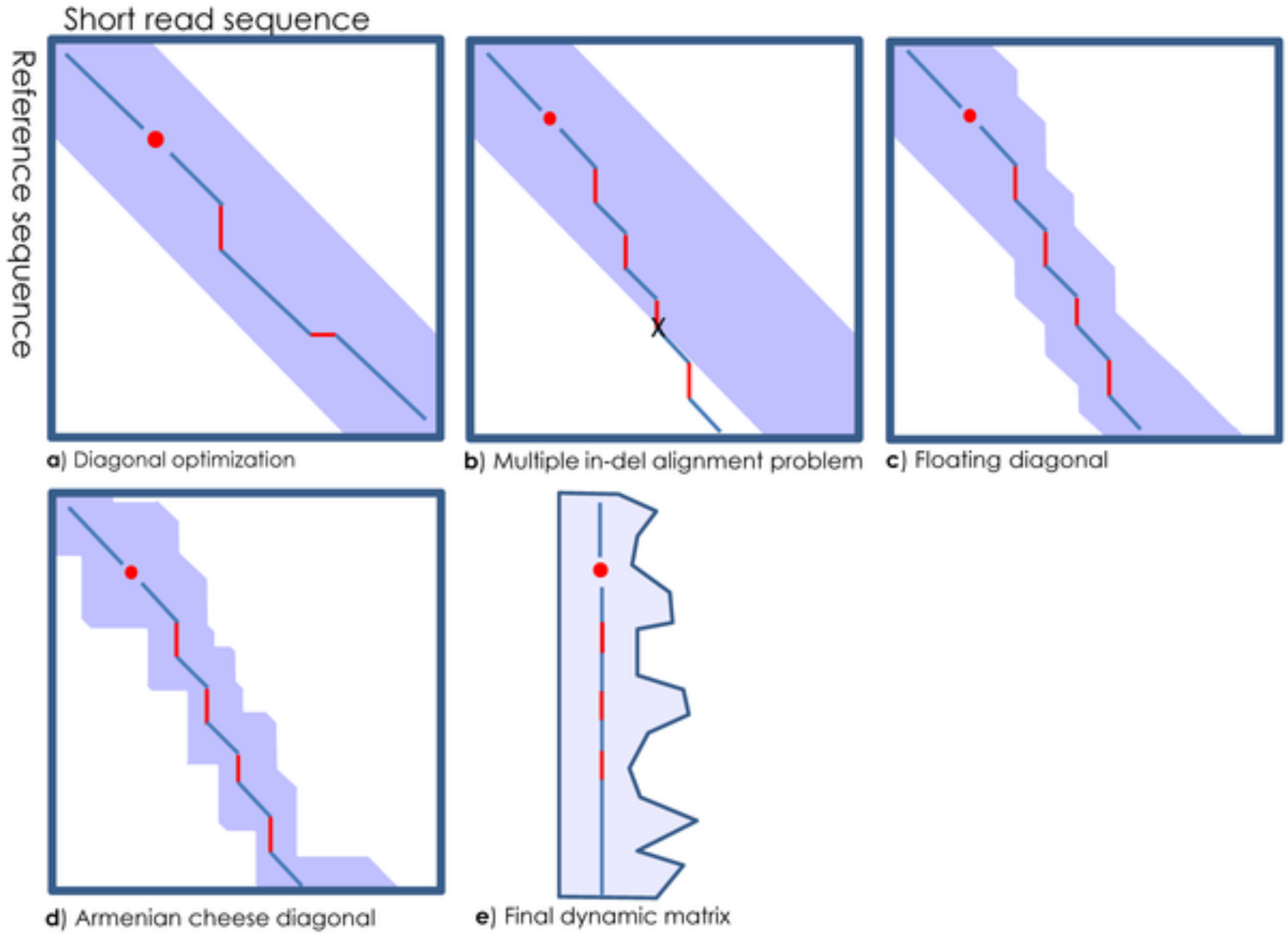
Reduction of complexity to the Needleman-Wunsch algorithm

- Calculating the entire DPA matrix seems wasteful, as values toward the off diagonal corners are necessarily high:
  - Why not cap the maximum cost and ignore areas of the matrix with high distance?
  - Why not partition the problem?
    Let a1, a2, …, an be partitions of input string a, and b1,.., bn of b. Then lev(a,b)<=lev(a1,b1)+…+lev(an,bn).



This idea leads to great savings, but not necessarily to the optimal alignment: The alignment is forced to go through certain points on the diagonal.

Reduction of complexity to the Needleman-Wunsch algorithm (2)



Santana-Quintero L, Dingerdissen H, Thierry-Mieg J, Mazumder R, Simonyan V (2014) HIVE-Hexagon: High-Performance, Parallelized Sequence Alignment for Next-Generation Sequencing Data Analysis. PLOS ONE 9(6): e99033. doi:10.1371/journal.pone.0099033
http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0099033

From global alignment to local alignment

- Sometimes we don't wish to compare all of both strings.
- We might want to see if one string is (approximately) contained in another:
  - Example: align 'wonder' with 'We had a wonderful time'

Distance *vs.* similarity

- Distance models the evolutionary process of change (sort of).
- Where sequences are the same length, relative distance is meaningful.
- Which is the more significant relationship?
  - Genes A and B, length 500, edit distance 5
  - Genes C and D, length 100, edit distance 5
- Similarity is sometimes more sensible.
- Similarity used in local alignment.

Local alignment

- Consider the following strings:
  - `TCGTAGTTTTGATCTT`
  - `TTTT`
- What will global alignment do?
- Indicate the overall difference in length.
  - Edit distance = 12

```
TCGTAGTTTTGATCTT
|  |  ||
T--T--TT--------
```

- We need local alignment.

Local alignment: naïve algorithm

- Take every substring $s_1[i]..s_1[j]$ in S_1 and align with every substring $s_2[i]..s_2[j]$ in S_2.
- Where $len(s_1)=m$, $len(s_2)=n$, what is the complexity?
  - How many substrings in S_1? in S_2?
  - How many pairs of substrings to align?
  - Cost of each alignment?
  - Complexity: O(   )

Distance *vs.* Similarity

$$D(i,j) = \min \begin{cases} \bullet D(i-1,j) + c_i & \textit{(insertion)} \\ \bullet D(i,j-1) + c_d & \textit{(deletion)} \\ \bullet D(i-1,j-1) + t(i,j) & \textit{(mis/match)} \end{cases}$$

$$S(i,j) = \max \begin{cases} \bullet S(i-1,j) + c_i & \textit{(insertion)} \\ \bullet S(i,j-1) + c_d & \textit{(deletion)} \\ \bullet S(i-1,j-1) + t(i,j) & \textit{(mis/match)} \end{cases}$$

Costs are different for distance and similarity.

Distance *vs.* Similarity: Costs

- Match:
  - Distance cost = 0
  - Similarity "cost" (reward) > 0
- Mismatch and indel:
  - Distance cost >0
  - Similarity "cost" (penalty) <0

Local alignment: Smith-Waterman algorithm

- $$S(i,j) = \max \begin{cases} \bullet S(i-1,j) + c_{indel} & \textit{(insertion)} \\ \bullet S(i,j-1) + c_{indel} & \textit{(deletion)} \\ \bullet S(i-1,j-1) + t(i,j) & \textit{((mis)/match)} \\ \bullet 0 \end{cases}$$

- Initialize: $S(i,0) = S(0,j) = 0$
- Cost (reward) for similarity (not distance):
  - Indel, mismatch costs <0
  - Match cost (benefit) >0
- Zero resets everytime costs outweigh benefits!
- T.F.Smith and M.Waterman, J.Mol.Biol. 147, 195-197, 1981
- T.F.Smith and M.Waterman, Adv. Appl. Math. 2, 482-489, 1981.

Local alignment: Smith-Waterman algorithm

- Finding alignments in the matrix:
  - Locate highest scoring cell(s)
  - Trace back to 0
- Example:
  - Using +1 for match, -1 for mismatch and indel
  - Local alignment of:
    - xyabcyz
    - prabcrs

Smith-Waterman local alignment

S(i,j)		p	r	a	b	p	r	a	b	c	r	s
	0	0	0	0	0	0	0	0	0	0	0	0
x	0											
y	0											
a	0											
b	0											
x	0											
y	0											
a	0											
b	0											
c	0											
x	0											
y	0											

Smith-Waterman local alignment

S(i,j)		p	r	a	b	p	r	a	b	c	r	s
	0	0	0	0	0	0	0	0	0	0	0	0
x	0	0	0	0	0	0	0	0	0	0	0	0
y	0	0	0	0	0	0	0	0	0	0	0	0
a	0	0	0	1	0	0	0	1	0	0	0	0
b	0											
x	0											
y	0											
a	0											
b	0											
c	0											
x	0											
y	0											

Smith-Waterman local alignment

S(i,j)		p	r	a	b	p	r	a	b	c	r	s
	0	0	0	0	0	0	0	0	0	0	0	0
x	0	0	0	0	0	0	0	0	0	0	0	0
y	0	0	0	0	0	0	0	0	0	0	0	0
a	0	0	0	1	0	0	0	1	0	0	0	0
b	0	0	0	0	2	1	0	0	2	1	0	0
x	0	0	0	0	1	1	0	0	1	1	0	0
y	0											
a	0											
b	0											
c	0											
x	0											
y	0											

Smith-Waterman local alignment

S(i,j)		p	r	a	b	p	r	a	b	c	r	s
	0	0	0	0	0	0	0	0	0	0	0	0
x	0	0	0	0	0	0	0	0	0	0	0	0
y	0	0	0	0	0	0	0	0	0	0	0	0
a	0	0	0	1	0	0	0	1	0	0	0	0
b	0	0	0	0	2	1	0	0	2	1	0	0
x	0	0	0	0	1	1	0	0	1	1	0	0
y	0	0	0	0	0	0	0	0	0	0	0	0
a	0	0	0	1	0	0	0	1	0	0	0	0
b	0	0	0	0	2	1	0	0	2	1	0	0
c	0	0	0	0	1	1	0	0	1	3	2	1
x	0											
y	0											

Smith-Waterman algorithm: complexity analysis

- Time complexity:

  $O(nm)$

- Space complexity:

  $O(nm)$

Alignment: a fine point

- Sequence similarity ≠ identity
- Sequence similarity ≠ homology

- Sequence similarity: approximately match
- Sequence homology:
  - Approximately match AND
  - Both sequences had the same ancestor.
- We often assume the common ancestor.

Smith-Waterman *vs.* Needleman-Wunsch

- Align the strings
  - legumes
  - legs

- Use Needleman-Wunsch global algorithm.
- Use Smith-Waterman local algorithm.

Multiple dispersed local alignments

- Multiple dispersed local alignments
  e.g.:
  - *S1: Bad girls have fun.*
  - *S2: The bad news for girls is that boys have all the fun.*
- Not catered for in NW or SW.
- Important biologically – exon or domain similarity.

Local alignment: multiple domains

- Lalign:
  - http://www.ch.embnet.org/software/LALIGN_form.html
  - Algorithm: X.Huang and W.Miller, "A time-efficient linear-space local similarity algorithm", *Advances in Applied Mathematics,* **12**: 337-357, 1991.

Fine points: Gaps

- Basic DPA model penalizes long gaps:
  - *c(gap of size 2k) = 2*c(gap of size k)*
- Affine gap: doesn't penalize long gaps as much:
  - Gap opening penalty *h (a large penalty)*
  - Gap extension penalty *g (a small penalty)*
  - *c(gap of size k) = h + (k-1)g*
  - *> c(gap of size 2k) = h + (2k-1)g << 2*c(gap of size k)*

SW/NW in the Context of Whole Genome Alignment

- Alignment is a rather computationally expensive operation on a whole-genome scale.
- Even with smart tricks such as banded alignments, the cost of NW-style alignments is still higher than a simple Hamming distance scan, which is too expensive already.
- We need more trickery yet:
  - We cannot avoid touching every read since they are all unique and different every time (every sequencing experiment).
  - However, the genome is always the same if you keep analysing the same organism (e.g. humans).
  - Leverage this fact to avoid scanning the entire reference genome for every single read!
  - Different types of index/tree/array data structures allow for this.

Solutions to the Alignment Complexity Issue

- Seed-and-extend techniques:
  - Instead of comparing the read to the entire genome, consider regions of similarity only:
    - Extract “seeds” from reads and find areas in the genome that share as many of these short sequences as possible (exact matching!)
    - A large pre-computed table of exact matching sequences allows for this to be done very efficiently (index: sequence -> list of positions in the genome).
- Clever data Structures:
  - BWT (https://en.wikipedia.org/wiki/Burrows-Wheeler_transform)
  - Suffix arrays…
  - (hash tables)
- The most popular aligners are:
  - Bowtie2 (Langmead B, Salzberg S. Fast gapped-read alignment with Bowtie 2. Nature Methods. 2012, 9:357-359.)
  - BWA (Li H, Durbin R (2009). "Fast and accurate short read alignment with Burrows–Wheeler Transform". Bioinformatics 25 (14): 1754–1760. doi:10.1093/bioinformatics/btp324. PMC 2705234. PMID 19451168)
  - Subread (Liao Y, Smyth GK and Shi W. The Subread aligner: fast, accurate and scalable read mapping by seed-and-vote. Nucleic Acids Research, 41(10):e108, 2013) – Developed at the WEHI Bioinformatics Division.

Summary

- Alignment is an essential technique in genomics…
  - … as it is the first step in line of data analysis for sequencing data.
  - It is also useful more generally (compare genomes, gene sequences, protein sequences…)
- Dynamic programming solutions somewhat ease the pain of determining edit distance, but are still prohibitively expensive on a whole genome scale giving rise to clever software engineering.

Aligner Parametrisation. Example: Bowtie2

```
…
Scoring:
  --ma <int>         match bonus (0 for --end-to-end, 2 for --local)
  --mp <int>         max penalty for mismatch; lower qual = lower penalty (6)
  --np <int>         penalty for non-A/C/G/Ts in read/ref (1)
  --rdg <int>,<int>  read gap open, extend penalties (5,3)
  --rfg <int>,<int>  reference gap open, extend penalties (5,3)
  --score-min <func> min acceptable alignment score w/r/t read length
                     (G,20,8 for local, L,-0.6,-0.6 for end-to-end)
 Paired-end:
  -I/--minins <int>  minimum fragment length (0)
  -X/--maxins <int>  maximum fragment length (500)
  --fr/--rf/--ff     -1, -2 mates align fw/rev, rev/fw, fw/fw (--fr)
  --no-mixed         suppress unpaired alignments for paired reads
  --no-discordant    suppress discordant alignments for paired reads
  --no-dovetail      not concordant when mates extend past each other
  --no-contain       not concordant when one mate alignment contains other
  --no-overlap       not concordant when mates overlap at all
```

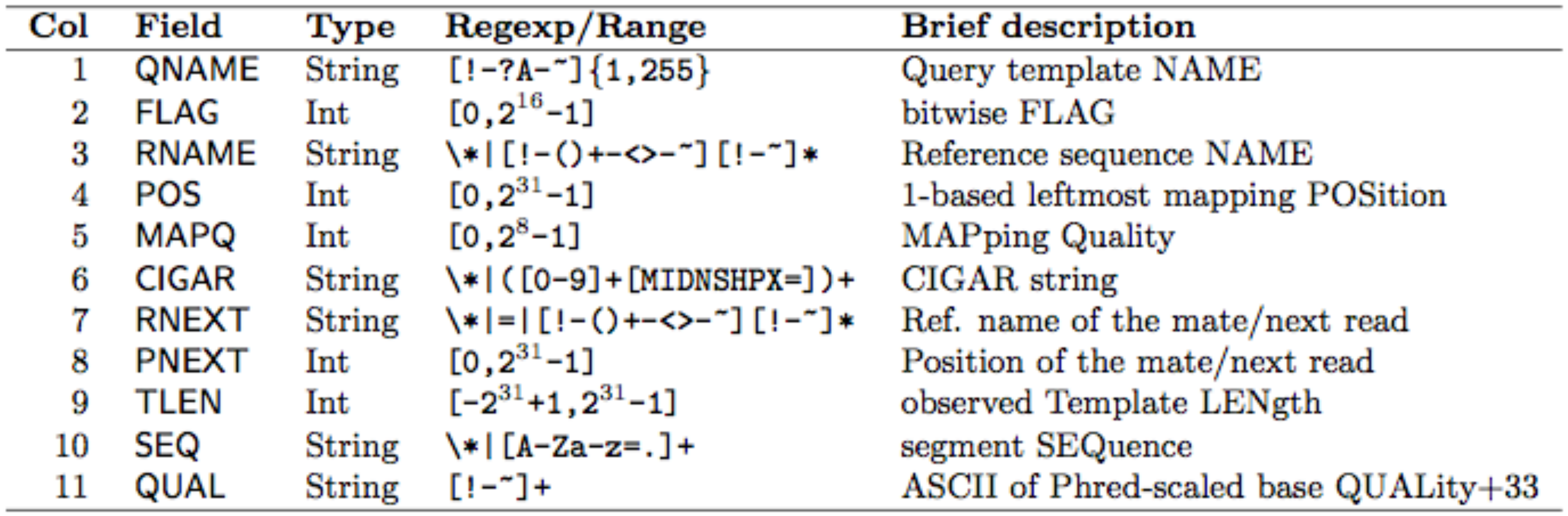
SAM Format

- SAM == Sequence Alignment/Map
- The Sequence Alignment/Map format and SAMtools.
  - Li H, Handsaker B, Wysoker A, Fennell T, Ruan J, Homer N, Marth G, Abecasis G, Durbin R; 1000 Genome Project Data Processing Subgroup.
  - Bioinformatics. 2009 Aug 15;25(16):2078-9. doi: 10.1093/bioinformatics/btp352. Epub 2009 Jun 8.
- The de facto standard for aligned read data.
- Organises data in a file: one read per line.
- Each line has at least 11 fields giving information about where the read maps and how well

SAM Read Information

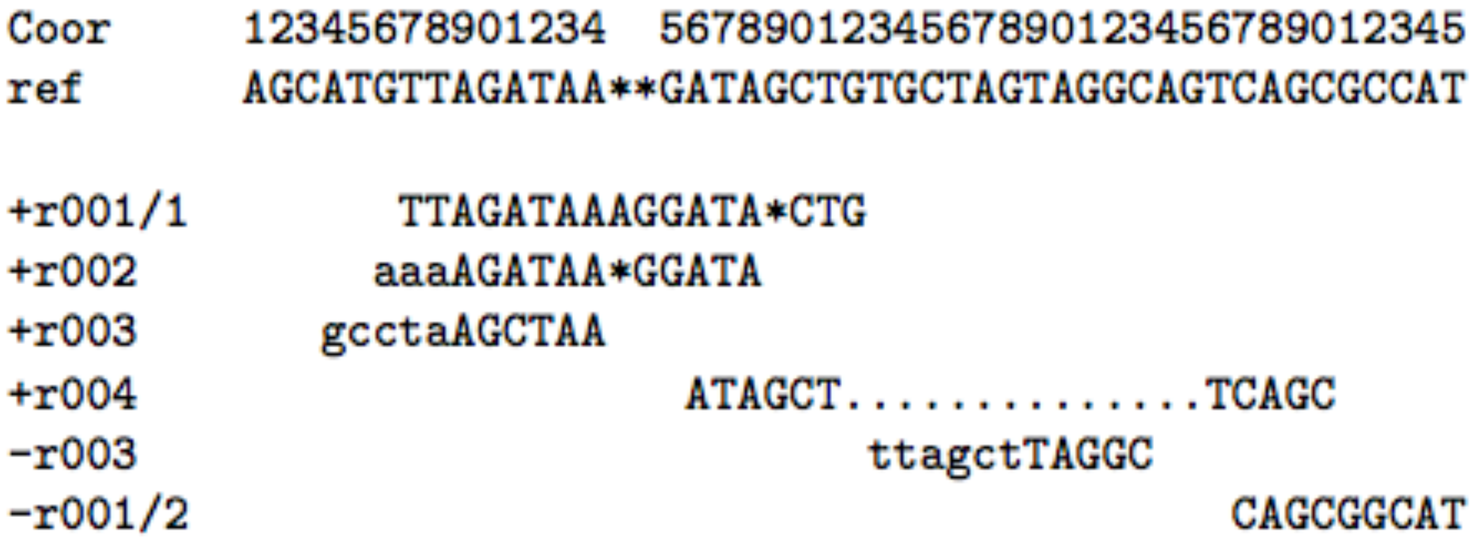
Everything SAMtools can be found here:

- http://samtools.github.io/hts-specs/SAMv1.pdf
- http://en.wikipedia.org/wiki/SAMtools



Col	Field	Type	Regexp/Range	Brief description
1	QNAME	String	`[!-?A-~]{1,255}`	Query template NAME
2	FLAG	Int	$[0,2^{16}-1]$	bitwise FLAG
3	RNAME	String	`\*|[!-()+-<>-~][!-~]*`	Reference sequence NAME
4	POS	Int	$[0,2^{31}-1]$	1-based leftmost mapping POSition
5	MAPQ	Int	$[0,2^{8}-1]$	MAPping Quality
6	CIGAR	String	`\*|([0-9]+[MIDNSHPX=])+`	CIGAR string
7	RNEXT	String	`\*|=|[!-()+-<>-~][!-~]*`	Ref. name of the mate/next read
8	PNEXT	Int	$[0,2^{31}-1]$	Position of the mate/next read
9	TLEN	Int	$[-2^{31}+1,2^{31}-1]$	observed Template LENgth
10	SEQ	String	`\*|[A-Za-z=.]+`	segment SEQuence
11	QUAL	String	`[!-~]+`	ASCII of Phred-scaled base QUALity+33

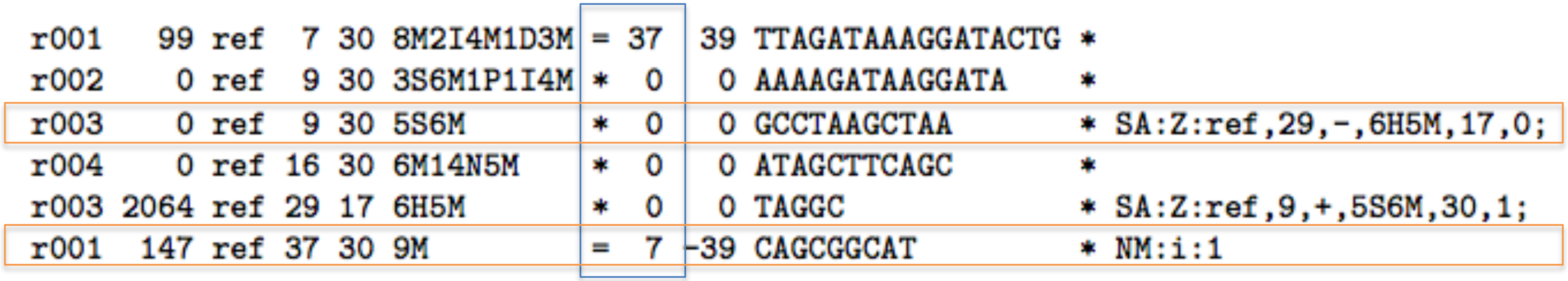
Alignment -> SAM Example



```
Coor      12345678901234  5678901234567890123456789012345
ref       AGCATGTTAGATAA**GATAGCTGTGCTAGTAGGCAGTCAGCGCCAT

+r001/1         TTAGATAAAGGATA*CTG
+r002          aaaAGATAA*GGATA
+r003        gcctaAGCTAA
+r004                    ATAGCT..............TCAGC
-r003                           ttagctTAGGC
-r001/2                                 CAGCGGCAT
```

Read name   Ref   Pos   Cigar   Mate Info   Read Sequence



```
r001   99 ref  7 30 8M2I4M1D3M = 37  39 TTAGATAAAGGATACTG *
r002    0 ref  9 30 3S6M1P1I4M *  0   0 AAAAGATAAGGATA    *
r003    0 ref  9 30 5S6M       *  0   0 GCCTAAGCTAA       * SA:Z:ref,29,-,6H5M,17,0;
r004    0 ref 16 30 6M14N5M    *  0   0 ATAGCTTCAGC       *
r003 2064 ref 29 17 6H5M       *  0   0 TAGGC             * SA:Z:ref,9,+,5S6M,30,1;
r001  147 ref 37 30 9M         =  7 -39 CAGCGGCAT         * NM:i:1
```

Example of SAM records

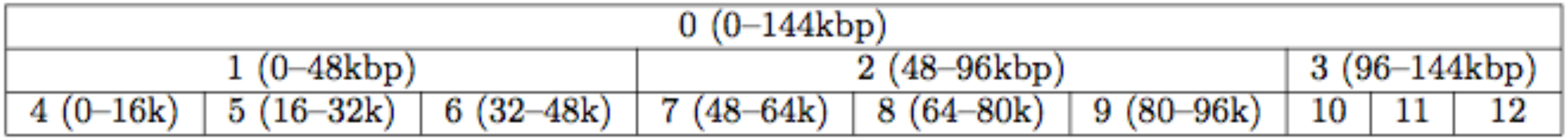
```
NS500643:117:HWLY2BGXX:1:22107:17460:11026 0 chr2L 9 0 85M * 0 0
CACGACAGAGGAAGCAGAACAGATATTTAGATTGCCTCTCATTTTCTCTCCCATATTATAGGGAGAAATATGATCGCGTATGCGA
AAAAAEEEEEEEEEEEEEEEEAEEEEEEEEEEAEEEEEEEEEEEEE/EEEEEEEAEEEEEEEEEAEEEEEEEEEEEEEE6AAAEEEAEE HI:i:1 NH:i:1 NM:i:0
NS500643:117:HWLY2BGXX:4:13406:25600:2160 0 chr2L 9 0 75M11S * 0 0
CACGACAGAGGAAGCAGAACAGATATTTAGATTGCCTCTCATTTTCTCTCCCATATTATAGGGAGAAATATGATCCTCGGCCGCGA
AAAAAEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEAEEEEEEEEEEEE6EEEEEEEEEEEEEEEEEE6AAEE/E<<AA HI:i:1 NH:i:1 NM:i:0
NS500643:117:HWLY2BGXX:1:22209:23968:8017 16 chr2L 9 0 75M * 0 0
CACGACAGGGGAAGCAGAACAGATATTTAGATTGCCTCTCATTTTCTCTCCCATATTATAGGGAGAAATATGATC
EAEEEE<EEAEEEAEEEEEEAEEEEEEEEEEA<AEEEEEEEA6EEEEEEEEEEE6EEEEE6EEAEEEEAEEEAEE6E HI:i:1 NH:i:1 NM:i:1
NS500643:117:HWLY2BGXX:1:11211:9064:4901 16 chr2L 10 0 74M * 0 0
ACGACAGAGGAAGCAGAACAGATATTTAGATTGCCTCTCATTTTCTCTCCCATATTATAGGGAGAAATATGATC
EEAEEEEEEEEE/EEEEEEEE<EEEAEAEEEEEAEEEAEEEEEEEEEEEAEAEEEEEEEEEEEEEEEEEEEEEEEE HI:i:1 NH:i:1 NM:i:0
NS500643:117:HWLY2BGXX:1:12109:26141:3294 16 chr2L 10 0 74M11S * 0 0
ACGACAGAGGAAGCAGAACAGATATTTAGATTGCCTCTCATTTTCTCTCCCATATTATAGGGAGAAATATGATCCTCGGCCGACC
EEAAEEEEE/EEEEEAEAEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE/EEEEEEEEEEAEEEEEEEAEEAAAAA HI:i:1 NH:i:1 NM:i:0
NS500643:117:HWLY2BGXX:1:13106:25024:18818 16 chr2L 10 0 74M * 0 0
ACGACAGAGGAAGCAGAACAGATATTTAGATTGCCTCTCATTTTCTCTCCCATATTATAGGGAGAAATATGATC
EA//EE<EEEEE<EEEEA/EEEEE6EEEAEE/AEE/EEEEEEE/EEEEEAAEEEEAEEEEEEEEEEAEAEEEEEEE HI:i:1 NH:i:1 NM:i:0
NS500643:117:HWLY2BGXX:1:13308:4737:5082 16 chr2L 10 0 74M * 0 0
ACGACAGAGGAAGCAGAACAGATATTTAGATTGCCTCTCATTTTCTCTCCCATATTATAGGGAGAAATATGATC
<AAEEEAE<AEEEEEEAEE/EEEEEEE<AEEEEE/EEEEEE6EEEEEEEEAEEEEEEEEEEEEEEEEEEEEEEEEE HI:i:1 NH:i:1 NM:i:0
NS500643:117:HWLY2BGXX:1:13312:20467:18158 16 chr2L 10 0 74M * 0 0
ACGACAGAGGAAGCAGAACAGATATTTAGATTGCCTCTCATTTTCTCTCCCATATTATAGGGAGAAATATGATC
/AEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEAEEEEEEE HI:i:1 NH:i:1 NM:i:0
NS500643:117:HWLY2BGXX:1:21110:11178:10992 16 chr2L 10 0 74M * 0 0
ACGACAGAGGAAGCAGAACAGATATTTAGATTGCCTCTCATTTTCTCTCCCATATTATAGGGAGAAATATGATC
A<EEEEEEAEE/EA/EEAEEEEEEEEEEEEEA</EEEEEEEEEAEE<EEEEEEEEE66EEEAEEEEEEEEEE<E/E HI:i:1 NH:i:1 NM:i:0
NS500643:117:HWLY2BGXX:1:21206:14654:12445 16 chr2L 10 0 74M * 0 0
ACGACAGAGGAAGCAGAACAGATATTTAGATTGCCTCTCATTTTCTCTCCCATATTATAGGGAGAAATATGATC
/EEEEEEE<EEEEEEAEEEEEEEEEEEEEEEEEEEEEAEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE HI:i:1 NH:i:1 NM:i:0
```

BAM Format

- BAM == Binary Alignment/Map
- Compressed SAM files for more efficient disk usage (BGZF).
- Thanks to redundant sequence, reference names, etc. the compression rate is usually around 50-95%.
- BAM organises the file in blocks of fixed size to enable semi-random access.

BAM Index (BAI)

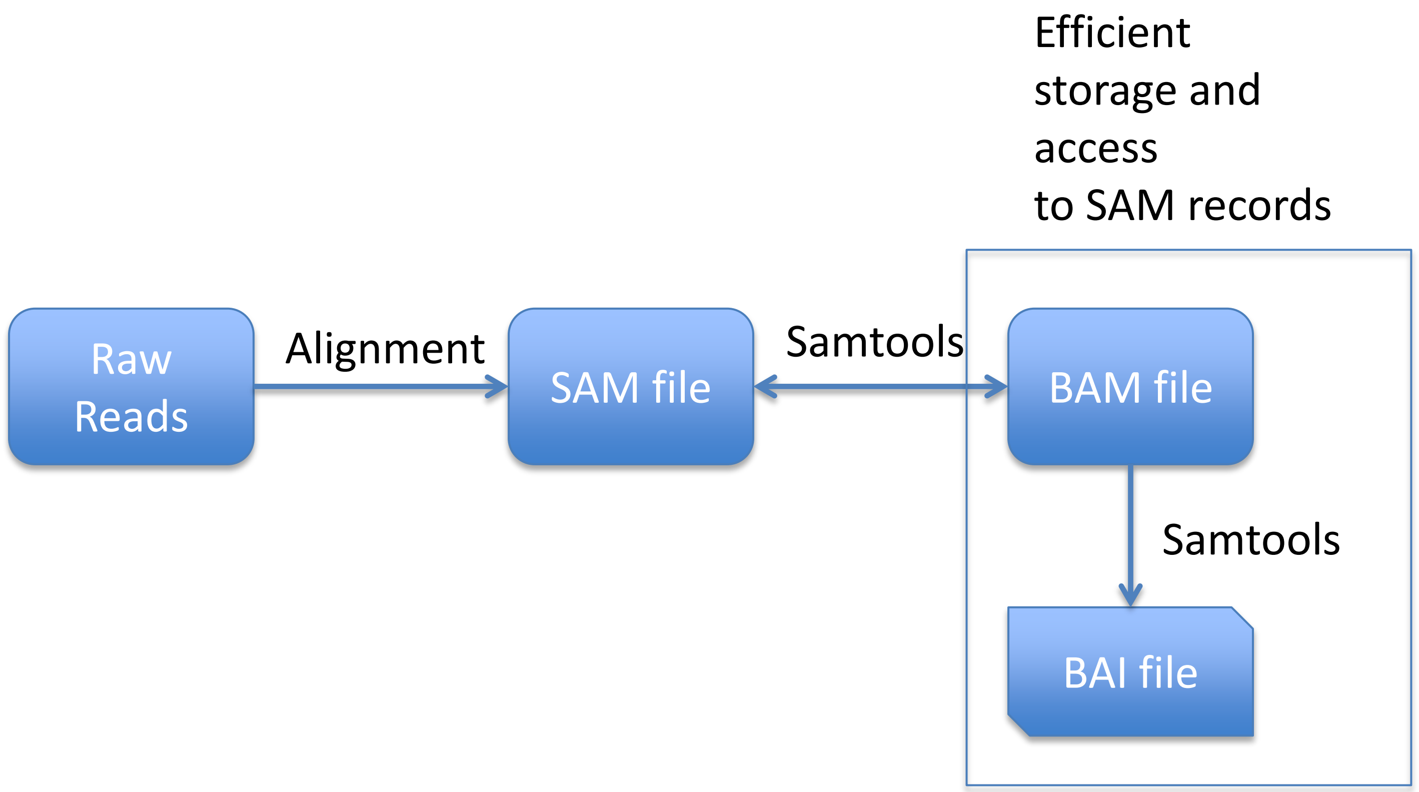
- (Sorted) BAM files can be indexed, to allow efficient access.



0 (0–144kbp)								
1 (0–48kbp)			2 (48–96kbp)			3 (96–144kbp)		
4 (0–16k)	5 (16–32k)	6 (32–48k)	7 (48–64k)	8 (64–80k)	9 (80–96k)	10	11	12

- Hierarchical bins allow to narrow down the region of the BAM file that contains a user query (e.g. an interval).
- The Index provides file-offsets to find the blocks that need to be decompressed to find the desired data.

SAM/BAM/BAI


Efficient storage and access to SAM records
Raw Reads
Alignment
SAM file
Samtools
BAM file
Samtools
BAI file

Samtools

Program:
samtools (Tools for alignments in the SAM format)

Version: 0.1.19-44428cd

Usage: samtools <command> [options]

Command:

view	SAM<->BAM conversion
sort	sort alignment file
mpileup	multi-way pileup
depth	compute the depth
faidx	index/extract FASTA
tview	text alignment viewer
index	index alignment
idxstats	BAM index stats (r595 or later)
fixmate	fix mate information
flagstat	simple stats
calmd	recalculate MD/NM tags and '=' bases
merge	merge sorted alignments
rmdup	remove PCR duplicates
reheader	replace BAM header
cat	concatenate BAMs
bedcov	read depth per BED region
targetcut	cut fosmid regions (for fosmid pool only)
phase	phase heterozygotes
bamshuf	shuffle and group alignments by name

Samtools: An Example Command

```
Usage:   samtools view [options] <in.bam>|<in.sam> [region1 [...]]

Options:
        -b            output BAM
        -h            print header for the SAM output
        -H            print header only (no alignments)
        -S            input is SAM
        -u            uncompressed BAM output (force -b)
        -1            fast compression (force -b)
        -x            output FLAG in HEX (samtools-C specific)
        -X            output FLAG in string (samtools-C specific)
        -c            print only the count of matching records
        -B            collapse the backward CIGAR operation
        -@ INT        number of BAM compression threads [0]
        -L FILE       output alignments overlapping the input BED FILE [null]
        -t FILE       list of reference names and lengths (force -S) [null]
        -T FILE       reference sequence file (force -S) [null]
        -o FILE       output file name [stdout]
        -R FILE       list of read groups to be outputted [null]
        -f INT        required flag, 0 for unset [0]
        -F INT        filtering flag, 0 for unset [0]
        -q INT        minimum mapping quality [0]
        -l STR        only output reads in library STR [null]
        -r STR        only output reads in read group STR [null]
        -s FLOAT      fraction of templates to subsample; integer part as seed [-1]
        -?            longer help
```

Used for SAM/BAM conversion:
samtools view –bS file.sam > file.bam
samtools view file.bam > file.sam

Used to extract reads in certain parts of the genome:
samtools view file.bam chr1:10-20