Combinatorial Optimisation in Bioinformatics

Introduction & Sequence Alignments

Yann Ponty · Sebastian Will

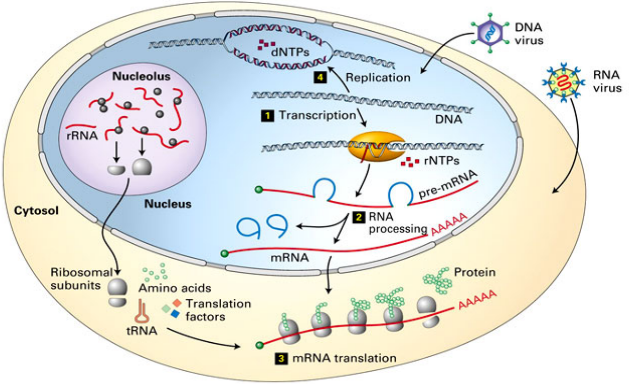
sebastian.will@polytechnique.edu · *yann.ponty@lix.polytechnique.fr*

The idea of this class

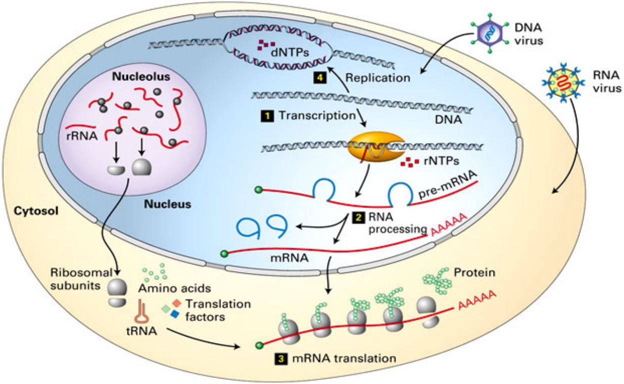
Why Combinatorial Optimization in Bioinformatics?

Bioinformatics is concerned with Molecular Biology

$$\overset{\textit{Replication}}{\circlearrowleft}\ \text{DNA} \xrightarrow{\textit{Transcription}} \text{RNA} \xrightarrow{\textit{Translation}} \text{Protein}$$



Bioinformatics is concerned with Molecular Biology

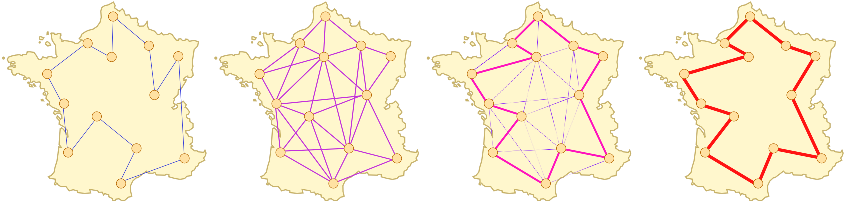


Where in this setting does computation make sense?
What can we learn (computationally)?

What is combinatorial optimization?

Example: Traveling Salesman

Problem: Given *n* cities, find shortest tour (round-trip)



- finite solution space (here: all city permutations)
- objective function (here: total distance)

What has Combinatorial Optimization to do with bioinformatics?

Typical biological problem: Find common sequence and structure motifs in the 5' regions of mRNAs that are upregulated under condition X over condition Y.

What has Combinatorial Optimization to do with bioinformatics?

Typical biological problem: Find common sequence and structure motifs in the 5' regions of mRNAs that are upregulated under condition X over condition Y.

Break down into subproblems:

- Determine upregulated genes
  - get (assembled) genome of your organism
  - sequence the mRNAs under conditions X and Y using NGS
  - map them to the genome (to measure expression level)
- compare 5' regions to identify common motifs
- predict RNA structures of 5' regions
- compare structures

What has Combinatorial Optimization to do with bioinformatics?

Typical biological problem: Find common sequence and structure motifs in the 5' regions of mRNAs that are upregulated under condition X over condition Y.

Break down into subproblems:

- Determine upregulated genes
  - get (assembled) genome of your organism
  - sequence the mRNAs under conditions X and Y using NGS
  - map them to the genome (to measure expression level)
- compare 5' regions to identify common motifs
- predict RNA structures of 5' regions
- compare structures

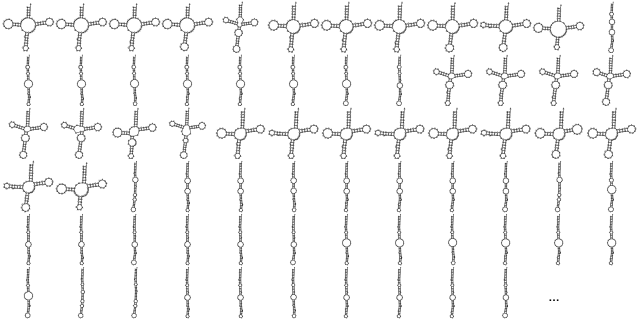
Finally, computational problems can be **formalized as combinatorial optimization problems.**

What is combinatorial optimization?

Example: RNA structure prediction

Formalize: 'Determine the best structure (out of all possible ones)'

GGGCUAUUAGCUCAGUUGGUUAGAGCGCACCCCUGAUAAGGGUGAGGUCGCUGAUUCGAAUUCAGCAUAGCCCA



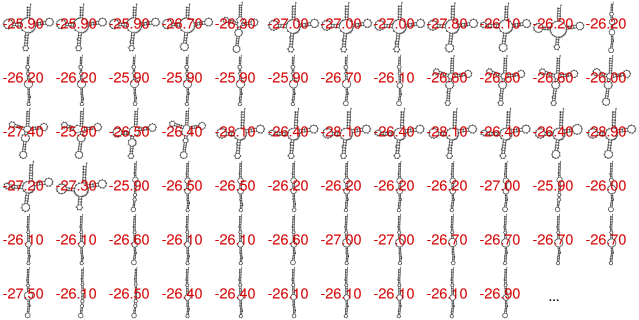
- finite solution space (here: RNA secondary structures)
- objective function (here: RNA energy function) ← energy model
- in which ways is it a typical example for CO in Bioinformatics?

What is combinatorial optimization?

Example: RNA structure prediction

Formalize: ’Determine the best structure (out of all possible ones)’

GGGCUAUUAGCUCAGUUGGUUAGAGCGCACCCCUGAUAAGGGUGAGGUCGCUGAUUCGAAUUCAGCAUAGCCCA

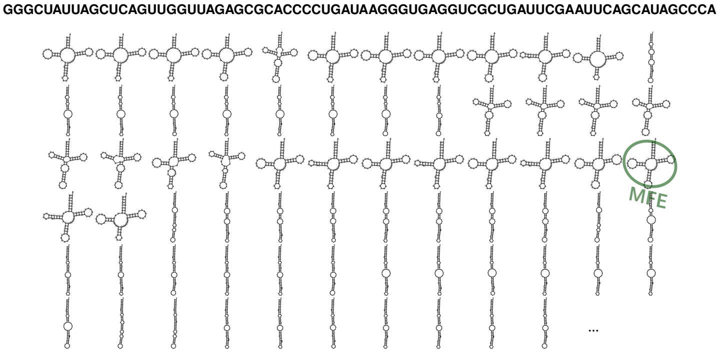


- finite solution space (here: RNA secondary structures)
- objective function (here: RNA energy function) ← energy model
- in which ways is it a typical example for CO in Bioinformatics?

What is combinatorial optimization?

Example: RNA structure prediction

Formalize: 'Determine the best structure (out of all possible ones)'



- finite solution space (here: RNA secondary structures)
- objective function (here: RNA energy function) ← energy model
- in which ways is it a typical example for CO in Bioinformatics?

Topics of the class(es)

- Jan 06th 2022 – YP: **Intro & Sequence Alignment**, Dynamic programming
- Jan 13th 2022 – SW: **Pattern Matching, Mapping**, Index data structures
- Jan 20th 2022 – YP: **Genome Assembly**, Graph algorithms
- Jan 27th 2022 – YP: **RNA structure prediction**, Dynamic programming
- Feb 03th 2022 – SW: **Advanced RNA structure prediction**
- Feb 10th 2022 – SW: **Comparative genomics**
- Feb 17th 2022 : **Exam**

Organisational stuff & grading

- Online Tools
    - Zoom, Slack (possibly Gather.Town, later)
    - Use Jupyter notebooks via Colab for programming: `https://colab.research.google.com/notebooks/intro.ipynb`
- Article presentations (60% of grade)
    - presentations in groups of three (with mixed backgrounds!)
    - each defense 15 mins (sharp!, 5min each) + 5-10 mins questions
    - we will let you choose from ∼10 articles (next week)
    - we plan presentations in classes of weeks 5 and 6
- Written exam: Feb, 17th (last class)

Class Topic: Sequence Alignment

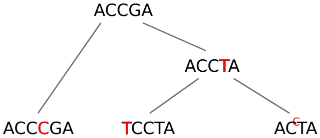
Sequence Alignment

Motivation: assess similarity of sequences and learn about their evolutionary relationship

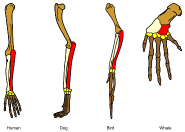
Example:

Sequences		*Alignment*
`ACCCGA`		`ACCCGA`
`ACTA`	$\Rightarrow_{\text{align}}$	`AC--TA`
`TCCTA`		`TCC-TA`

Homology: Alignment reasonable, if sequences homologous



Two (or more sequences) are *homologous* if they evolved from a common ancestor.
Homology inherited by letters through *correspondences* induced by columns



[Homology in anatomy]

Plan: from simple pairwise to multiple alignment

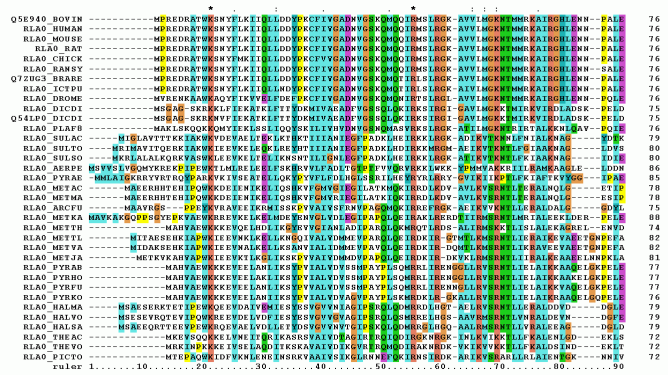
- **pairwise alignment**

```
Sequence A: ACGTGAACT
Sequence B: AGTGAGT
```

⇒align A and B

```
ACGTGAACT
A-GTGA-GT
```

- **variants**: global and local, realistic gap costs, . . .
- **multiple alignment**



```
                                 *   .      :       .          .   *         : : :      .
Q5E940_BOVIN -----------MPREDRATWKSNYFLKIIQLLDDYPKCFIVGADNVGSKQMQQIRMSLRGK-AVVLMGKNTMMRKAIRGHLENN--PALE   76
 RLA0_HUMAN -----------MPREDRATWKSNYFLKIIQLLDDYPKCFIVGADNVGSKQMQQIRMSLRGK-AVVLMGKNTMMRKAIRGHLENN--PALE   76
 RLA0_MOUSE -----------MPREDRATWKSNYFLKIIQLLDDYPKCFIVGADNVGSKQMQQIRMSLRGK-AVVLMGKNTMMRKAIRGHLENN--PALE   76
   RLA0_RAT -----------MPREDRATWKSNYFLKIIQLLDDYPKCFIVGADNVGSKQMQQIRMSLRGK-AVVLMGKNTMMRKAIRGHLENN--PALE   76
 RLA0_CHICK -----------MPREDRATWKSNYFMKIIQLLDDYPKCFVVGADNVGSKQMQQIRMSLRGK-AVVLMGKNTMMRKAIRGHLENN--PALE   76
 RLA0_RANSY -----------MPREDRATWKSNYFLKIIQLLDDYPKCFIVGADNVGSKQMQQIRMSLRGK-AVVLMGKNTMMRKAIRGHLENN--SALE   76
Q7ZUG3_BRARE -----------MPREDRATWKSNYFLKIIQLLDDYPKCFIVGADNVGSKQMQTIRLSLRGK-AVVLMGKNTMMRKAIRGHLENN--PALE   76
 RLA0_ICTPU -----------MPREDRATWKSNYFLKIIQLLNDYPKCFIVGADNVGSKQMQTIRLSLRGK-AIVLMGKNTMMRKAIRGHLENN--PALE   76
 RLA0_DROME -----------MVRENKAAWKAQYFIKVVELFDEFPKCFIVGADNVGSKQMQNIRTSLRGL-AVVLMGKNTMMRKAIRGHLENN--PQLE   76
 RLA0_DICDI -----------MSGAG-SKRKKLFIEKATKLFTTYDKMIVAEADFVGSSQLQKIRKSIRGI-GAVLMGKKTMIRKVIRDLADSK--PELD   75
Q54LP0_DICDI -----------MSGAG-SKRKNVFIEKATKLFTTYDKMIVAEADFVGSSQLQKIRKSIRGI-GAVLMGKKTMIRKVIRDLADSK--PELD   75
 RLA0_PLAF8 -----------MAKLSKQQKKQMYIEKLSSLIQQYSKILIVHVDNVGSNQMASVRKSLRGK-ATILMGKNTRIRTALKKNLQAV--PQIE   76
 RLA0_SULAC -----MIGLAVTTTKKIAKWKVDEVAELTEKLKTHKTIIIANIEGFPADKLHEIRKKLRGK-ADIKVTKNNLFNIALKNAG-----YDTK   79
 RLA0_SULTO ----MRIMAVITQERKIAKWKIEEVKELEQKLREYHTIIIANIEGFPADKLHDIRKKMRGM-AEIKVTKNTLFGIAAKNAG-----LDVS   80
 RLA0_SULSO ----MKRLALALKQRKVASWKLEEVKELTELIKNSNTILIGNLEGFPADKLHEIRKKLRGK-ATIKVTKNTLFKIAAKNAG-----IDIE   80
 RLA0_AERPE MSVVSLVGQMYKREKPIPEWKTLMLRELEELFSKHRVVLFADLTGTPTFVVQRVRKKLWKK-YPMMVAKKRIILRAMKAAGLE---LDDN   86
 RLA0_PYRAE -MMLAIGKRRYVRTRQYPARKVKIVSEATELLQKYPYVFLFDLHGLSSRILHEYRYRLRRY-GVIKIIKPTLFKIAFTKVYGG---IPAE   85
 RLA0_METAC ------MAEERHHTEHIPQWKKDEIENIKELIQSHKVFGMVGIEGILATKMQKIRRDLKDV-AVLKVSRNTLTERALNQLG-----ETIP   78
 RLA0_METMA ------MAEERHHTEHIPQWKKDEIENIKELIQSHKVFGMVRIEGILATKIQKIRRDLKDV-AVLKVSRNTLTERALNQLG-----ESIP   78
 RLA0_ARCFU ------MAAVRGS---PPEYKVRAVEEIKRMISSKPVVAIVSFRNVPAGQMQKIRREFRGK-AEIKVVKNTLLERALDALG-----GDYL   75
 RLA0_METKA MAVKAKGQPPSGYEPKVAEWKRREVKELKELMDEYENVGLVDLEGIPAPQLQEIRAKLRERDTIIRMSRNTLMRIALEEKLDER--PELE   88
 RLA0_METTH -------------MAHVAEWKKKEVQELHDLIKGYEVVGIANLADIPARQLQKMRQTLRDS-ALIRMSKKTLISLALEKAGREL--ENVD   74
 RLA0_METTL -------MITAESEHKIAPWKIEEVNKLKELLKNGQIVALVDMMEVPARQLQEIRDKIR-GTMTLKMSRNTLIERAIKEVAEETGNPEFA   82
 RLA0_METVA -------MIDAKSEHKIAPWKIEEVNALKELLKSANVIALIDMMEVPAVQLQEIRDKIR-DQMTLKMSRNTLIKRAVEEVAEETGNPEFA   82
 RLA0_METJA --------METKVKAHVAPWKIEEVKTLKGLIKSKPVVAIVDMMDVPAPQLQEIRDKIR-DKVKLRMSRNTLIIRALKEAAEELNNPKLA   81
 RLA0_PYRAB -------------MAHVAEWKKKEVEELANLIKSYPVIALVDVSSMPAYPLSQMRRLIRENGGLLRVSRNTLIELAIKKAAQELGKPELE   77
 RLA0_PYRHO -------------MAHVAEWKKKEVEELAKLIKSYPVIALVDVSSMPAYPLSQMRRLIRENGGLLRVSRNTLIELAIKKAAKELGKPELE   77
 RLA0_PYRFU -------------MAHVAEWKKKEVEELANLIKSYPVVALVDVSSMPAYPLSQMRRLIRENNGLLRVSRNTLIELAIKKVAQELGKPELE   77
 RLA0_PYRKO -------------MAHVAEWKKKEVEELANIIKSYPVIALVDVAGVPAYPLSKMRDKLR-GKALLRVSRNTLIELAIKRAAQELGQPELE   76
 RLA0_HALMA -----MSAESERKTETIPEWKQEEVDAIVEMIESYESVGVVNIAGIPSRQLQDMRRDLHGT-AELRVSRNTLLERALDDVD-----DGLE   79
 RLA0_HALVO -----MSESEVRQTEVIPQWKREEVDELVDFIESYESVGVVGVAGIPSRQLQSMRRELHGS-AAVRMSRNTLVNRALDEVN-----DGFE   79
 RLA0_HALSA -----MSAEEQRTTEEVPEWKRQEVAELVDLLETYDSVGVVNVTGIPSKQLQDMRRGLHGQ-AALRMSRNTLLVRALEEAG-----DGLD   79
 RLA0_THEAC -------------MKEVSQQKKELVNEITQRIKASRSVAIVDTAGIRTRQIQDIRGKNRGK-INLKVIKKTLLFKALENLGD----EKLS   72
 RLA0_THEVO -------------MRKINPKKKEIVSELAQDITKSKAVAIVDIKGVRTRQMQDIRAKNRDK-VKIKVVKKTLLFKALDSIND----EKLT   72
 RLA0_PICTO -------------MTEPAQWKIDFVKNLENEINSRKVAAIVSIKGLRNNEFQKIRNSIRDK-ARIKVSRARLLRLAIENTGK----NNIV   72
      ruler 1.......10........20........30........40........50........60........70........80........90
```

A first attempt to compare sequences: Levenshtein Distance

Sequences are words over alphabets Σ, e.g. $\Sigma = \{A, C, G, T\}$.

Definition

The *Levenshtein Distance* between two words/sequences is the minimal number of substitutions, insertions and deletions to transform one into the other.

Example

ACCCGA and ACTA have (at most) distance 3:
ACCCGA $\rightarrow$ ACCGA $\rightarrow$ ACCTA $\rightarrow$ ACTA

In biology, operations have different cost. (Why?)

Edit Distance: Operations

Definition (Edit Operations)

An *edit operation* is a pair $(x, y) \in (\Sigma \cup \{-\} \neq (-, -)$. We call (x,y)

- *substitution* iff $x \neq -$ and $y \neq -$
- *deletion* iff $y = -$
- *insertion* iff $x = -$

For sequences a, b, write $a \rightarrow_{(x,y)} b$, iff a is transformed to b by operation (x, y).
Furthermore, write $a \Rightarrow_S b$, iff a is transformed to b by a sequence of edit operations S.

Example

ACCCGA $\rightarrow_{(C,-)}$ ACCGA $\rightarrow_{(G,T)}$ ACCTA $\rightarrow_{(-,T)}$ ATCCTA

ACCCGA $\Rightarrow_{(C,-),(G,T),(-,T)}$ ATCCTA

Recall: $- \notin \Sigma$, a, b are sequences in Σ^*

Edit Distance: Cost and Problem Definition

Definition (Cost, Edit Distance)

Let w be a cost function on edit operations, then the *edit distance of sequences a and b* is the minimum cost of all sequences S of edit operations that transform a to b.

Edit Distance: Cost and Problem Definition

Definition (Cost, Edit Distance)

Let w be a cost function on edit operations, then the *edit distance of sequences a and b* is the minimum cost of all sequences S of edit operations that transform a to b.

Remarks

- Does it match our idea of evolution?
- Is it Combinatorial Optimization?
- How to compute edit distance efficiently? not at all obvious $\Rightarrow$ alignments

Alignments

Example

$a = \texttt{ACGGAT}$
$b = \texttt{CCGCTT}$

possible alignments are

$$\begin{array}{l} \hat{a} = \texttt{AC-GG-AT} \\ \hat{b} = \texttt{-CCGCT-T} \end{array} \quad \text{or} \quad \begin{array}{l} \hat{a} = \texttt{ACGG---AT} \\ \hat{b} = \texttt{--CCGCT-T} \end{array} \quad \text{or} \ldots \text{(exponentially many)}$$

edit operations of first alignment: (A,-),(-,C),(G,C),(-,T),(A,-)

Definition (Alignment)

A pair of words $\hat{a}, \hat{b} \in (\Sigma \cup \{-\})^*$ is called *alignment of sequences a and b* ($\hat{a}$ and $\hat{b}$ are called *alignment strings*), iff

1. $|\hat{a}| = |\hat{b}|$
2. for all $1 \leq i \leq |\hat{a}|$: $\hat{a}_i \neq -$ or $\hat{b}_i \neq -$
3. deleting all gap symbols $-$ from $\hat{a}$ yields a
   and deleting all $-$ from $\hat{b}$ yields b

Best alignment distance = best edit distance

The columns of an alignment $(\hat{a}, \hat{b})$ correspond to edit operations; we score it by adding the cost of these operations.

$$\sum_{i=1}^{|\hat{a}|} w(\hat{a}_i, \hat{b}_i)$$

The best alignment distance equals the best edit distance (if the cost of edit operations is a metric).

Best alignment distance = best edit distance

The columns of an alignment $(\hat{a}, \hat{b})$ correspond to edit operations; we score it by adding the cost of these operations.

$$\sum_{i=1}^{|\hat{a}|} w(\hat{a}_i, \hat{b}_i)$$

The best alignment distance equals the best edit distance (if the cost of edit operations is a metric).

What is the significance of this?

- Edit distance is biologically well motivated, but there is no obvious way to efficiently optimize it.
- Alignment distance is equivalent.
- → focus on alignments. One can optimize over these combinatorial objects efficiently.

Derive best alignments from smaller best alignments

Example

a = `CACGGCT`
b = `CCGCTG`

The best alignment ends in either

- (`T`,`G`); we get it from the best alignment of (the prefixes) `CACGGC` and `CCGCT`,
- or (`T`,-); we get it from aligning `CACGGC` and *b*,
- or (-,`G`); we get it from aligning *a* and `CCGCT`.

This recursive decomposition strategy is possible because the problem has the property of 'optimal substructure': "the prefix alignment of any optimal alignment is itself optimal".

Remarks

- does this immediately allow us to optimize efficiently?
- the property allows us to apply dynamic programming
- many problems in bioinformatics have this property

Recursion of the Needleman-Wunsch Algorithm

Define a function D(i,j), to compute the (best) alignment distance for the prefix sequences $a_{1..i}$ and $b_{1..j}$.
$D(i,j)$ can be implemented by based on the decomposition idea of the last slide:

$$D(i,j) = \min \begin{cases} D(i-1,j-1) + w(a_i,b_j) & (\mathit{match}) \\ D(i-1,j) + w(a_i,-) & (\mathit{deletion}) \\ D(i,j-1) + w(-,b_j) & (\mathit{insertion}) \end{cases}$$

This works only for $i > 0$ and $j > 0$, in these special cases

- $D(0,0) = 0$
- $D(i,0) = D(i-1,0) + w(a_i,-)$
- $D(0,j) = D(0,j-1) + w(-,b_j)$

Let's code it!

Recursion alone, does not allow for efficient computation, because of overlapping subproblems!

Recursion + Alignment Matrix: the Needleman-Wunsch Algorithm

To evaluate the recursion efficiently, use a matrix to store all partial solutions $D(i, j)$.
The *alignment matrix* of a and b is the $(n+1) \times (m+1)$-matrix that contains at each entry (i, j) the alignment distances of the prefixes $a_{1..i}$ and $b_{1..j}$.

Example

$a = \texttt{AT}$, $b = \texttt{AAGT}$; $w(x, y) = \begin{cases} 0 & \text{iff } x = y \\ 1 & \text{otherwise} \end{cases}$

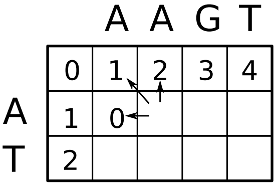
		A	A	G	T
A					
T					

Recursion + Alignment Matrix: the Needleman-Wunsch Algorithm

To evaluate the recursion efficiently, use a matrix to store all partial solutions $D(i,j)$.
The *alignment matrix* of a and b is the $(n+1)\times(m+1)$-matrix that contains at each entry (i,j) the alignment distances of the prefixes $a_{1..i}$ and $b_{1..j}$.

Example

$$a = \texttt{AT},\ b = \texttt{AAGT}\ ;\ w(x,y) = \begin{cases} 0 & \text{iff } x = y \\ 1 & \text{otherwise} \end{cases}$$



$$\begin{array}{c|ccccc} & & A & A & G & T \\ \hline & 0 & 1 & 2 & 3 & 4 \\ A & 1 & 0 & & & \\ T & 2 & & & & \end{array}$$

Recursion + Alignment Matrix: the Needleman-Wunsch Algorithm

To evaluate the recursion efficiently, use a matrix to store all partial solutions $D(i,j)$.
The *alignment matrix* of a and b is the $(n+1)\times(m+1)$-matrix that contains at each entry (i,j) the alignment distances of the prefixes $a_{1..i}$ and $b_{1..j}$.

Example

$$a = \mathtt{AT},\ b = \mathtt{AAGT}\ ;\ w(x,y) = \begin{cases} 0 & \text{iff } x = y \\ 1 & \text{otherwise} \end{cases}$$

		A	A	G	T
	0	1	2	3	4
A	1	0	1	2	3
T	2	1	1	2	2

Let's code it!

How to find the best aligmnment?

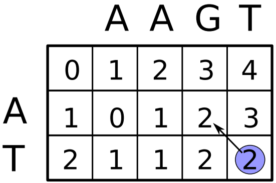
Example

- $a = \texttt{AT}$, $b = \texttt{AAGT}$
- $w(x,y) = \begin{cases} 0 & \text{iff } x = y \\ 1 & \text{otherwise} \end{cases}$

		A	A	G	T
	0	1	2	3	4
A	1	0	1	2	3
T	2	1	1	2	2

Traceback

$$w(x,y) = \begin{cases} 0 & \text{iff } x = y \\ 1 & \text{otherwise} \end{cases}$$



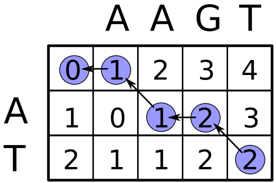
		A	A	G	T
	0	1	2	3	4
A	1	0	1	2	3
T	2	1	1	2	2

Remarks

- Start in (n, m). For every (i, j) determine optimal case.
- Not necessarily unique.
- Sequence of *trace arrows* let's infer best alignment.

Traceback

$$w(x, y) = \begin{cases} 0 & \text{iff } x = y \\ 1 & \text{otherwise} \end{cases}$$



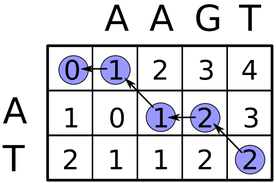
		A	A	G	T
	0	1	2	3	4
A	1	0	1	2	3
T	2	1	1	2	2

Remarks

- Start in (n, m). For every (i, j) determine optimal case.
- Not necessarily unique.
- Sequence of *trace arrows* let's infer best alignment.

Traceback

$$w(x,y) = \begin{cases} 0 & \text{iff } x = y \\ 1 & \text{otherwise} \end{cases}$$



		A	A	G	T
	0	1	2	3	4
A	1	0	1	2	3
T	2	1	1	2	2

Remarks

- Start in (n,m). For every (i,j) determine optimal case.
- Not necessarily unique.
- Sequence of *trace arrows* let's infer best alignment.

Let's code it!

Complexity

- compute one entry: three cases, i.e. constant time
- nm entries $\Rightarrow$ fill matrix in $O(nm)$ time
- traceback: $O(n+m)$ time
- **Overall:** $O(n^2)$ time and space (assuming $m \leq n$)

Remarks

- assuming $m \leq n$ is w.l.o.g. since we can exchange a and b
- space complexity can be improved to $O(n)$ for computation of distance (simple, “store only current and last row”) and traceback (more involved; Hirschberg-algorithm uses “Divide and Conquer” for computing trace)

Plan

- We have seen how to compute the pairwise edit distance and the corresponding optimal alignment.
- Before going multiple, we will look at two further special topics for pairwise alignment:
    - more realistic, non-linear gap cost and
    - similarity scores and local alignment

Alignment Cost Revisited

Motivation:

- The alignments $\begin{array}{c}\texttt{GA--T}\\\texttt{GAAGT}\end{array}$ and $\begin{array}{c}\texttt{G-A-T}\\\texttt{GAAGT}\end{array}$ have the same edit distance.
- The first one is biologically more reasonable: it is more likely that evolution introduces one large gap than two small ones.
- This means: gap cost should be non-linear, sub-additive!

Gap Penalty

A *gap penalty* is a function $g : \mathbb{N} \to \mathbb{R}$ that is sub-additive, i.e.

$$g(k+l) \leq g(k) + g(l).$$

A *gap* in an alignment string $\hat{a}$ is a substring of $\hat{a}$ that consists of only gap symbols $-$ and is maximally extended. $\Delta^{\hat{a}}$ is the multi-set of gaps in $\hat{a}$.

The *alignment cost with gap penalty g of $(\hat{a}, \hat{b})$* is

$$\begin{aligned} w_g(\hat{a}, \hat{b}) = & \sum_{\substack{1 \leq r \leq |\hat{a}|, \\ \text{where } \hat{a}_r \neq -, \hat{b}_r \neq -}} w(\hat{a}_r, \hat{b}_r) && (\textit{cost of mismatchs}) \\ & + \sum_{x \in \Delta^{\hat{a}} \uplus \Delta^{\hat{b}}} g(|x|) && (\textit{cost of gaps}) \end{aligned}$$

Example:

$$\begin{aligned} \hat{a} &= \texttt{ATG---CGAC--GC} \quad \Rightarrow \Delta^{\hat{a}} = \{\texttt{---}, \texttt{--}\}, \; \Delta^{\hat{b}} = \{\texttt{-}, \texttt{-}\} \\ \hat{b} &= \texttt{-TGCGGCG-CTTTC} \end{aligned}$$

General sub-additive gap penalty

Let D be the alignment matrix of a and b with cost w and gap penalty g, such that $D_{i,j} = w_g(a_{1..i}, b_{1..j})$. Then:

- $D_{0,0} = 0$
- for all $1 \leq i \leq n$: $D_{i,0} = g(i)$
- for all $1 \leq j \leq m$: $D_{0,j} = g(j)$
- $$D_{i,j} = \min \begin{cases} D_{i-1,j-1} + w(a_i, b_j) & (\textit{match}) \\ \min_{1\leq k\leq i} D_{i-k,j} + g(k) & (\textit{deletion of length } k) \\ \min_{1\leq k\leq j} D_{i,j-k} + g(k) & (\textit{insertion of length } k) \end{cases}$$

Remarks

- Complexity $O(n^3)$ time, $O(n^2)$ space
- pseudocode, correctness, traceback left as exercise
- much more realistic, but significantly more expensive than Needleman-Wunsch
  $\Rightarrow$ can we improve it?

General sub-additive gap penalty

Let D be the alignment matrix of a and b with cost w and gap penalty g, such that $D_{i,j} = w_g(a_{1..i}, b_{1..j})$. Then:

- $D_{0,0} = 0$
- for all $1 \leq i \leq n$: $D_{i,0} = g(i)$
- for all $1 \leq j \leq m$: $D_{0,j} = g(j)$
- $D_{i,j} = \min \begin{cases} D_{i-1,j-1} + w(a_i, b_j) & (\textit{match}) \\ \min_{1 \leq k \leq i} D_{i-k,j} + g(k) & (\textit{deletion of length } k) \\ \min_{1 \leq k \leq j} D_{i,j-k} + g(k) & (\textit{insertion of length } k) \end{cases}$

Remarks

- Complexity $O(n^3)$ time, $O(n^2)$ space
- pseudocode, correctness, traceback left as exercise
- much more realistic, but significantly more expensive than Needleman-Wunsch

$\Rightarrow$ can we improve it?

Let's code it!?

Affine gap cost

Definition

A gap penalty is affine, iff there are real constants α and β, such that for all $k \in \mathbb{N}$:
$g(k) = \alpha + \beta k$.

Remarks

- Affine gap penalties almost as good as general ones: Distinguishing gap opening (α) and gap extension cost (β) is “biologically reasonable”.
- The minimal alignment cost with affine gap penalty can be computed in $O(n^2)$ time! (Gotoh algorithm)

Gotoh algorithm

In addition to the alignment matrix D, define two further matrices/states.

- $A_{i,j}$:= cost of best alignment of $a_{1..i}, b_{1..j}$, that ends with deletion $\overset{a_i}{\underset{-}{|}}$.
- $B_{i,j}$:= cost of best alignment of $a_{1..i}, b_{1..j}$, that ends with insertion $\overset{-}{\underset{b_j}{|}}$.

Recursions:

$$A_{i,j} = \min \begin{cases} A_{i-1,j} + \beta & (\textit{deletion extension}) \\ D_{i-1,j} + g(1) & (\textit{deletion opening}) \end{cases}$$

$$B_{i,j} = \min \begin{cases} B_{i,j-1} + \beta & (\textit{insertion extension}) \\ D_{i,j-1} + g(1) & (\textit{insertion opening}) \end{cases}$$

$$D_{i,j} = \min \begin{cases} D_{i-1,j-1} + w(a_i, b_j) & (\textit{match}) \\ A_{i,j} & (\textit{deletion closing}) \\ B_{i,j} & (\textit{insertion closing}) \end{cases}$$

Remark: $O(n^2)$ time and space

Gotoh algorithm

In addition to the alignment matrix D, define two further matrices/states.

- $A_{i,j}$:= cost of best alignment of $a_{1..i}, b_{1..j}$, that ends with deletion $\begin{smallmatrix} a_i \\ | \\ - \end{smallmatrix}$.
- $B_{i,j}$:= cost of best alignment of $a_{1..i}, b_{1..j}$, that ends with insertion $\begin{smallmatrix} - \\ | \\ b_j \end{smallmatrix}$.

Recursions:

$$A_{i,j} = \min \begin{cases} A_{i-1,j} + \beta & (\textit{deletion extension}) \\ D_{i-1,j} + g(1) & (\textit{deletion opening}) \end{cases}$$

$$B_{i,j} = \min \begin{cases} B_{i,j-1} + \beta & (\textit{insertion extension}) \\ D_{i,j-1} + g(1) & (\textit{insertion opening}) \end{cases}$$

$$D_{i,j} = \min \begin{cases} D_{i-1,j-1} + w(a_i, b_j) & (\textit{match}) \\ A_{i,j} & (\textit{deletion closing}) \\ B_{i,j} & (\textit{insertion closing}) \end{cases}$$

Remark: $O(n^2)$ time and space

Let's code it!

Similarity

The similarity of an alignment $(\hat{a}, \hat{b})$ is $s(\hat{a}, \hat{b}) = \sum_{i=1}^{|\hat{a}|} s(\hat{a}_i, \hat{b}_i)$, where $s : (\Sigma \cup \{-\})^2 \to \mathbb{R}$ is a *similarity function* $\big(s(x, x) > 0,\ s(x, -) < 0,\ s(-, x) < 0\big)$.

Observation. Instead of minimizing alignment cost, one can maximize similarity:

$$S_{i,j} = \max \begin{cases} S_{i-1,j-1} + s(a_i, b_j) \\ S_{i-1,j} + s(a_i, -) \\ S_{i,j-1} + s(-, b_j) \end{cases}$$

Why similarity?

- Defining similarity of 'building blocks' is often more natural; easier to learn.
- Similarity is useful for *local alignment*

Local Alignment Motivation

Local alignment asks for the best alignment of any two subsequences of *a* and *b*.
Important Application: Search!
(e.g. BLAST combines heuristics and local alignment)

Example

$a = \texttt{AWGVIAC}\underline{\texttt{AILAG}}\texttt{RS}$
$b = \texttt{VIVT}\underline{\texttt{AIAVAG}}\texttt{YY}$

In contrast, all previous methods compute “global alignments”.
Why is distance not useful?

Example

a) `XXXAAXXXX` / `YYAAYY` b) `XXAAAAAXXXX` / `YYYAAAAAYY`

Where is the stronger local motif? Only similarity can distinguish.

Local Alignment

Definition (Local Alignment Problem)

Let s be a similarity on alignments.

$$S_{\text{global}}(a,b) := \max_{\substack{(\hat{a},\hat{b})\\ \text{alignment of } a \text{ and } b}} s(\hat{a},\hat{b}) \qquad (\textit{global similarity})$$

$$S_{\text{local}}(a,b) := \max_{\substack{1\leq i'<i\leq n\\ 1\leq j'<j\leq m}} S_{\text{global}}(a_{i'..i}, b_{j'..j}) \qquad (\textit{local similarity})$$

The *local alignment problem* is to compute $S_{\text{local}}(a,b)$.

Remarks

- That is, local alignment asks for the subsequences of a and b that have the best alignment.
- How would we define the local alignment matrix for DP?
- Case in point, why does "$H_{i,j} := S_{\text{local}}(a_{1..i}, b_{1..j})$" not work?

Local Alignment Matrix

The *local alignment matrix* H of a and b is the $n+1 \times m+1$ matrix of entries

$$H_{i,j} := \max_{0\leq i'\leq i, 0\leq j'\leq j} S_{\text{global}}(a_{i'+1..i}, b_{j'+1..j}).$$

Remarks

- $S_{\text{local}}(a,b) = \max_{i,j} H_{i,j}$ (!)
- all entries $H_{i,j} \geq 0$, since $S_{\text{global}}(\epsilon,\epsilon) = 0$.
- $H_{i,j} = 0$ implies no (non-empty) subsequences of a and b that end in respective i and j are similar.
- Allows case distinction / optimal substructure property holds.

Local Alignment Algorithm — Case Distinction

Cases for $H_{i,j}$

$$\text{1.)}\quad \begin{array}{c|c} \ldots & a_i \\ \ldots & b_i \end{array} \qquad \text{2.)}\quad \begin{array}{c|c} \ldots & a_i \\ \ldots & - \end{array} \qquad \text{3.)}\quad \begin{array}{c|c} \ldots & - \\ \ldots & b_j \end{array}$$

4.) 0, since if each of the above cases is dissimilar (i.e. negative), there is still (ϵ, ϵ).

Local Alignment Algorithm (Smith-Waterman Algorithm)

For the local alignment matrix H of *a* and *b*,

- $H_{0,0} = 0$
- for all $1 \leq i \leq n$: $H_{i,0} = 0$
- for all $1 \leq j \leq m$: $H_{0,j} = 0$
- $$H_{i,j} = \max \begin{cases} 0 & (\textit{empty alignment}) \\ H_{i-1,j-1} + s(a_i, b_j) & \\ H_{i-1,j} + s(a_i, -) & \\ H_{i,j-1} + s(-, b_j) & \end{cases}$$

Local Alignment Algorithm (Smith-Waterman Algorithm)

For the local alignment matrix H of *a* and *b*,

- $H_{0,0} = 0$
- for all $1 \leq i \leq n$: $H_{i,0} = 0$
- for all $1 \leq j \leq m$: $H_{0,j} = 0$
- $$H_{i,j} = \max \begin{cases} 0 & (\textit{empty alignment}) \\ H_{i-1,j-1} + s(a_i, b_j) \\ H_{i-1,j} + s(a_i, -) \\ H_{i,j-1} + s(-, b_j) \end{cases}$$

Let's code it!

Local Alignment Remarks

Remarks

- Complexity $O(n^2)$ time and space, again space complexity can be improved
- Requires that similarity function is centered around zero, i.e. positive = similar, negative = dissimilar.
- Extension to affine gap cost works
- Traceback?

Local Alignment Example

Example

- $a = \texttt{AAC}$, $b = \texttt{ACAA}$
- $s(x, y) = \begin{cases} 2 & \text{iff } x = y \\ -3 & \text{otherwise} \end{cases}$

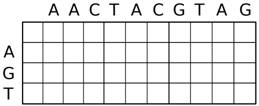
		A	C	A	A
	0	0	0	0	0
A	0	2	0	2	2
A	0	2	0	2	4
C	0	0	4	1	1

Traceback: start at maximum entry, trace back to first 0 entry

Exercise / Homework: semi-local “glocal” alignemnt

- Also known as free end-gap alignment.
- Case in point, align a short sequence *a* to a subsequence of a long(er) sequence *b*. Leave gaps at the beginning and end of *b* free of cost.

```
- - - - A - G T - -
A A C T A C G T A G
```



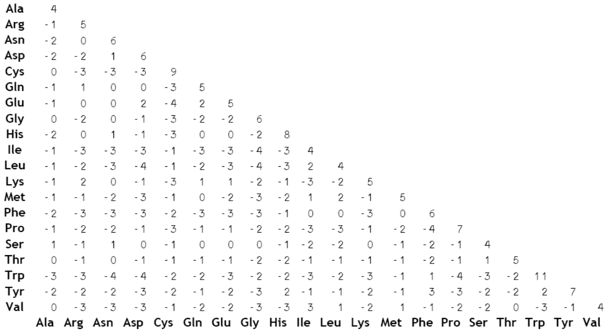
Green end-gaps are free

- How would you modify your implementation of Smith-Waterman? (code it!)
- Analogous variants make costs free at the beginning and/or end of *a*.
- Can you imagine, where such algorithms are useful?

Substitution/Similarity Matrices

- In practice: use similarity matrices learned from closely related sequences or multiple alignments
- PAM (Percent Accepted Mutations) for proteins
- BLOSUM (BLOcks of Amino Acid SUbstitution) for proteins
- RIBOSUM for RNA
- Scores are (scaled) log odd scores: $log \frac{Pr[x,y|\text{ Related}]}{Pr[x,y|\text{ Background}]}$

Case in point, BLOSUM62:



Ala	4																			
Arg	-1	5																		
Asn	-2	0	6																	
Asp	-2	-2	1	6																
Cys	0	-3	-3	-3	9															
Gln	-1	1	0	0	-3	5														
Glu	-1	0	0	2	-4	2	5													
Gly	0	-2	0	-1	-3	-2	-2	6												
His	-2	0	1	-1	-3	0	0	-2	8											
Ile	-1	-3	-3	-3	-1	-3	-3	-4	-3	4										
Leu	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4									
Lys	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5								
Met	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5							
Phe	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6						
Pro	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7					
Ser	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4				
Thr	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5			
Trp	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11		
Tyr	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	
Val	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4
	Ala	**Arg**	**Asn**	**Asp**	**Cys**	**Gln**	**Glu**	**Gly**	**His**	**Ile**	**Leu**	**Lys**	**Met**	**Phe**	**Pro**	**Ser**	**Thr**	**Trp**	**Tyr**	**Val**

Multiple Alignment

Example: *Sequences* *Alignment*

$$\begin{array}{rcl} a^{(1)} & = & \texttt{ACCCGAG} \\ a^{(2)} & = & \texttt{ACTACC} \\ a^{(3)} & = & \texttt{TCCTACGG} \end{array} \quad \Rightarrow_{\text{align}} \quad A = \begin{array}{l} \texttt{ACCCGA-G-} \\ \texttt{AC--TAC-C} \\ \texttt{TCC-TACGG} \end{array}$$

Definition

A *multiple alignment A of K sequences $a^{(1)}...a^{(K)}$* is a $K \times N$-matrix $(A_{i,j})_{\substack{1\leq i\leq K \\ 1\leq j\leq N}}$ (N is the *number of columns of A*)

where

1. each entry $A_{i,j} \in (\Sigma \cup \{-\})$
2. for each row i: deleting all gaps from $(A_{i,1}...A_{i,N})$ yields $a^{(i)}$
3. no column j contains only gap symbols

How to Score Multiple Alignments

As for pairwise alignment:

- Assume columns are scored independently
- Score is sum over alignment columns

$$S(A) = \sum_{j=1}^{N} s(A_{1j}, \ldots, A_{Kj})$$

Example

$$S(A) = s\begin{pmatrix} A \\ A \\ T \end{pmatrix} + s\begin{pmatrix} C \\ C \\ C \end{pmatrix} + s\begin{pmatrix} C \\ - \\ C \end{pmatrix} + s\begin{pmatrix} C \\ - \\ - \end{pmatrix} + \cdots + s\begin{pmatrix} - \\ C \\ G \end{pmatrix}$$

How do we know similarities?

How to Score Multiple Alignments

As for pairwise alignment:

- Assume columns are scored independently
- Score is sum over alignment columns

$$S(A) = \sum_{j=1}^{N} s\begin{pmatrix}A_{1j}\\ \cdots\\ A_{Kj}\end{pmatrix}$$

Example

$$S(A) = s\begin{pmatrix}A\\A\\T\end{pmatrix} + s\begin{pmatrix}C\\C\\C\end{pmatrix} + s\begin{pmatrix}C\\-\\C\end{pmatrix} + s\begin{pmatrix}C\\-\\-\end{pmatrix} + \cdots + s\begin{pmatrix}-\\C\\G\end{pmatrix}$$

How to define $s\begin{pmatrix}x\\y\\z\end{pmatrix}$? as log odds $s\begin{pmatrix}x\\y\\z\end{pmatrix} = log\frac{Pr[x,y,z|\text{ Related}]}{Pr[x,y,z|\text{ Background}]}$?

Problems? Can we learn similarities for triples, 4-tuples, ...?

Sum-Of-Pairs Score

Idea: approximate column scores by pairwise scores

$$s\begin{pmatrix} x_1 \\ \dots \\ x_j \end{pmatrix} = \sum_{1\leq k<l\leq K} s(x_k, x_l)$$

Sum-of-pairs is the most commonly used scoring scheme for multiple alignments.
(Extensible to gap penalties, in particular affine gap cost)

Drawbacks?

Optimal Multiple Alignment

Idea: use dynamic programming

Example

For 3 sequences a, b, c, use 3-dimensional matrix (after initialization:)

$$S_{i,j,k} = \max \begin{cases} S_{i-1,j-1,k-1} & +s(a_i, b_j, c_k) \\ S_{i-1,j-1,k} & +s(a_i, b_j, -) \\ S_{i-1,j,k-1} & +s(a_i, -, c_k) \\ S_{i,j-1,k-1} & +s(-, b_j, c_k) \\ S_{i-1,j,k} & +s(a_i, -, -) \\ S_{i,j-1,k} & +s(-, b_j, -) \\ S_{i,j,k-1} & +s(-, -, c_k) \end{cases}$$

For K sequences use K-dimensional matrix.

Complexity?

Heuristic Multiple Alignment: Progressive Alignment

Idea: compute optimal alignments only pairwise

Example

4 sequences $a^{(1)}, a^{(2)}, a^{(3)}, a^{(4)}$

1. determine how they are related
   $\Rightarrow$ tree, e.g. $((a^{(1)}, a^{(2)}), (a^{(3)}, a^{(4)}))$
2. align most closely related sequences first
   $\Rightarrow$ (optimally) align $a^{(1)}$ and $a^{(2)}$ by DP
3. go on $\Rightarrow$ (optimally) align $a^{(3)}$ and $a^{(4)}$ by DP
4. go on?! $\Rightarrow$ (optimally) align the two alignments
   How can we do that?
5. Done. We produced a multiple alignment of $a^{(1)}, a^{(2)}, a^{(3)}, a^{(4)}$.

Remarks: - Optimality is not guaranteed. *Why?*
- The tree is known as guide tree. *How can we get it?*

Guide tree

The guide tree determines the order of pairwise alignments in the progressive alignment scheme.
The order of the progressive alignment steps is crucial for quality!

Heuristics:

1. Compute pairwise distances between all input sequences
    - align all against all
    - in case, transform similarities to distances (e.g. Feng-Doolittle)
2. Cluster sequences by their distances, e.g. by
    - Unweighted Pair Group Method (UPGMA)
    - Neighbor Joining (NJ)

Aligning Alignments

Two (multiple) alignments A and B can be aligned by DP (like two sequences).

Idea:

- An alignment is a sequence of alignment columns.

  Example: $$\begin{array}{l}\texttt{ACCCGA-G-}\\ \texttt{AC--TAC-C}\\ \texttt{TCC-TACGG}\end{array} \equiv \begin{pmatrix}A\\A\\T\end{pmatrix}\begin{pmatrix}C\\C\\C\end{pmatrix}\begin{pmatrix}C\\-\\C\end{pmatrix}\begin{pmatrix}C\\-\\-\end{pmatrix}\dots\begin{pmatrix}-\\C\\G\end{pmatrix}.$$

- Assign similarity to two columns from A and B, e.g. $s(\begin{pmatrix}-\\C\\G\end{pmatrix}, \begin{pmatrix}G\\C\end{pmatrix})$ by *sum-of-pairs*.

Apply dynamic programming (recurse over alignment scores of prefixes of alignments)

Consequences for progressive alignment scheme:

- Optimization only *local*.
- Commits to local decisions. “Once a gap, always a gap”

Aligning Alignments

Two (multiple) alignments A and B can be aligned by DP (like two sequences).

Idea:

- An alignment is a sequence of alignment columns.

Example:

```
ACCCGA-G-
AC--TAC-C
TCC-TACGG
```

$$\equiv \begin{pmatrix}A\\A\\T\end{pmatrix}\begin{pmatrix}C\\C\\C\end{pmatrix}\begin{pmatrix}C\\-\\C\end{pmatrix}\begin{pmatrix}C\\-\\-\end{pmatrix}\ldots\begin{pmatrix}-\\C\\G\end{pmatrix}.$$

- Assign similarity to two columns from A and B, e.g. $s(\begin{pmatrix}-\\C\\G\end{pmatrix}, \begin{pmatrix}G\\C\end{pmatrix})$ by *sum-of-pairs*.

Apply dynamic programming (recurse over alignment scores of prefixes of alignments)

Consequences for progressive alignment scheme:

- Optimization only *local*.
- Commits to local decisions. “Once a gap, always a gap”

Let’s code it! ???

Progressive Alignment — Example

$$\text{IN: } a^{(1)} = ACCG, a^{(2)} = TTGG, a^{(3)} = TCG, a^{(4)} = CTGG \; w(x,y) = \begin{cases} 0 \text{ iff } x = y \\ 2 \text{ iff } x = - \text{ or } y = - \\ 3 \text{ otherwise (for mismatch)} \end{cases}$$

- Compute all against all edit distances and cluster

Align ACCG and TTGG

		T	T	G	G
	0	2	4	6	8
A	2	3	5	7	9
C	4	5	6	8	10
C	6	7	8	9	11
G	8	9	10	8	9

Align ACCG and TCG

		T	C	G
	0	2	4	6
A	2	3	5	7
C	4	5	3	6
C	6	7	5	6
G	8	9	8	5

Align ACCG and CTGG

		C	T	G	G
	0	2	4	6	8
A	2	3	5	7	9
C	4	2	5	8	10
C	6	4	5	8	11
G	8	7	7	5	8

Align TTGG and TCG

		T	C	G
	0	2	4	6
T	2	0	3	6
T	4	2	3	6
G	6	5	5	3
G	8	8	8	5

Align TTGG and CTGG

		C	T	G	G
	0	2	4	6	8
T	2	3	2	5	8
T	4	5	3	5	8
G	6	7	6	3	5
G	8	9	9	6	3

Align TCG and CTGG

		C	T	G	G
	0	2	4	6	8
T	2	3	2	5	8
C	4	2	5	5	8
G	6	5	5	5	5

Progressive Alignment — Example

IN: $a^{(1)} = ACCG, a^{(2)} = TTGG, a^{(3)} = TCG, a^{(4)} = CTGG$ $w(x, y) = \begin{cases} 0 \text{ iff } x = y \\ 2 \text{ iff } x = - \text{ or } y = - \\ 3 \text{ otherwise (for mismatch)} \end{cases}$

- Compute all against all edit distances and cluster

  $\Rightarrow$ distance matrix

$$\begin{array}{ccccc} & a^{(1)} & a^{(2)} & a^{(3)} & a^{(4)} \\ a^{(1)} & 0 & 9 & 5 & 8 \\ a^{(2)} & & 0 & 5 & 3 \\ a^{(3)} & & & 0 & 5 \\ a^{(4)} & & & & 0 \end{array}$$

  $\Rightarrow$ Cluster (e.g. UPGMA)
  $a^{(2)}$ and $a^{(4)}$ are closest, Then: $a^{(1)}$ and $a^{(3)}$

Progressive Alignment — Example

IN: $a^{(1)} = ACCG, a^{(2)} = TTGG, a^{(3)} = TCG, a^{(4)} = CTGG$ $w(x,y) = \begin{cases} 0 \text{ iff } x = y \\ 2 \text{ iff } x = - \text{ or } y = - \\ 3 \text{ otherwise (for mismatch)} \end{cases}$

- Compute all against all edit distances and cluster
  $\Rightarrow$ guide tree $((a^{(2)}, a^{(4)}), (a^{(1)}, a^{(3)}))$
- Align $a^{(2)}$ and $a^{(4)}$: $\begin{array}{l}\texttt{TTGG}\\ \texttt{CTGG}\end{array}$, Align $a^{(1)}$ and $a^{(3)}$: $\begin{array}{l}\texttt{ACCG}\\ \texttt{-TCG}\end{array}$
- Align the alignments!

Align $\begin{array}{l}\texttt{TTGG}\\ \texttt{CTGG}\end{array}$ and $\begin{array}{l}\texttt{ACCG}\\ \texttt{-TCG}\end{array}$

		A -	C T	C C	G G
	0	4	12	20	28
TC	8	10	. . .		
TT	16	⋮	⋱		
GG	24				
GG	32				

Progressive Alignment — Example

IN: $a^{(1)} = ACCG, a^{(2)} = TTGG, a^{(3)} = TCG, a^{(4)} = CTGG$ $w(x, y) = \begin{cases} 0 \text{ iff } x = y \\ 2 \text{ iff } x = - \text{ or } y = - \\ 3 \text{ otherwise (for mismatch)} \end{cases}$

- Compute all against all edit distances and cluster
  $\Rightarrow$ guide tree $((a^{(2)}, a^{(4)}), (a^{(1)}, a^{(3)}))$
- Align $a^{(2)}$ and $a^{(4)}$: $\begin{array}{l}\texttt{TTGG}\\ \texttt{CTGG}\end{array}$, Align $a^{(1)}$ and $a^{(3)}$: $\begin{array}{l}\texttt{ACCG}\\ \texttt{-TCG}\end{array}$
- Align the alignments!

Align $\begin{array}{l}\texttt{TTGG}\\ \texttt{CTGG}\end{array}$ and $\begin{array}{l}\texttt{ACCG}\\ \texttt{-TCG}\end{array}$

		A -	C T	C C	G G
	0	4	12	20	28
TC	8	10	. . .		
TT	16	⋮	⋱		
GG	24				
GG	32				

- $w(TC, --) = w(T, -) + w(C, -) + w(T, -) + w(C, -) = 8$
- $w(--, A-) = w(-, A) + w(-, -) + w(-, A) + w(-, -) = 4$
- $w(TC, A-) = w(T, A) + w(C, A) + w(T, -) + w(C, -) = 10$
- $w(TC, CT) = w(T, C) + w(C, C) + w(T, T) + w(C, T) = 6$
- $\ldots$

Progressive Alignment — Example

$$\text{IN: } a^{(1)} = ACCG, a^{(2)} = TTGG, a^{(3)} = TCG, a^{(4)} = CTGG \quad w(x,y) = \begin{cases} 0 \text{ iff } x = y \\ 2 \text{ iff } x = - \text{ or } y = - \\ 3 \text{ otherwise (for mismatch)} \end{cases}$$

- Compute all against all edit distances and cluster
  $\Rightarrow$ guide tree $((a^{(2)}, a^{(4)}), (a^{(1)}, a^{(3)}))$
- Align $a^{(2)}$ and $a^{(4)}$: $\begin{array}{l}\texttt{TTGG}\\\texttt{CTGG}\end{array}$, Align $a^{(1)}$ and $a^{(3)}$: $\begin{array}{l}\texttt{ACCG}\\\texttt{-TCG}\end{array}$
- Align the alignments!

Align $\begin{array}{l}\texttt{TTGG}\\\texttt{CTGG}\end{array}$ and $\begin{array}{l}\texttt{ACCG}\\\texttt{-TCG}\end{array}$

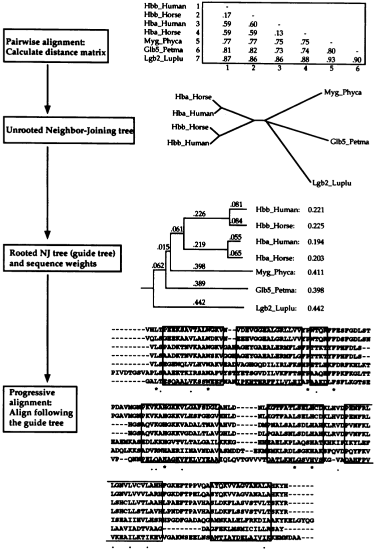
		A -	C T	C C	G G
	0	4	12	20	28
TC	8	10	…		
TT	16	⋮	⋱		
GG	24				
GG	32				

$\Longrightarrow$

after filling
and traceback

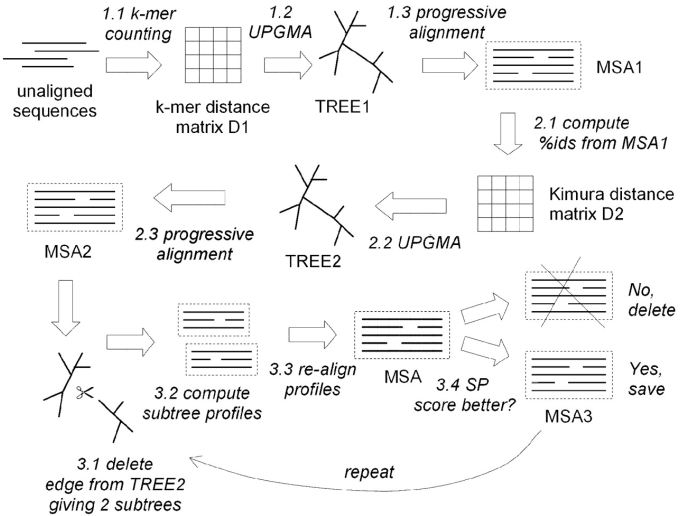
```
TTGG
CTGG
ACCG
-TCG
```

A Classical Approach: `CLUSTALW`



- prototypical progressive alignment
- similarity score with affine gap cost
- neighbor joining for tree construction
- special ‘tricks’ for gap handling

Advanced Progressive Alignment in `MUSCLE`



unaligned sequences

1.1 k-mer counting

k-mer distance matrix D1

1.2 UPGMA

TREE1

1.3 progressive alignment

MSA1

2.1 compute %ids from MSA1

Kimura distance matrix D2

2.2 UPGMA

TREE2

2.3 progressive alignment

MSA2

3.1 delete edge from TREE2 giving 2 subtrees

3.2 compute subtree profiles

3.3 re-align profiles

MSA

3.4 SP score better?

No, delete

Yes, save

MSA3

repeat

1.) alignment draft and 2.) reestimation 3.) iterative refinement