Multiple Sequence Alignment

CMSC 423
Carl Kingsford

jazz_timeline.js (~/Subversion/prj/jazzmap/prototype3-protovis/js) - VIM

```
+-- 41 lines: Foreword here --------------------------
       labelColor: "white",
       labelFont: "14px sans-serif",
       backgroundColor: "black",
       dotBarStyle: 'green',//'rgba(255, 128, 128,
       dotFillStyle: 'rgba(150,150,150,0.7)',
       dotStrokeStyle: "rgb(100, 100, 100)",
       dotHighlightFill: "rgb(255, 237, 160)",
       dotHighlightStroke: "rgba(100,100,100,0.3)"
       dotSelectFill: "coral",
       dotSelectStroke: "rgba(100,100,100,0.3)",
       dotBorderWidth: 1,
       dotBarPaddingLeft: 0, // 10
       dotBarPaddingRight: 0, // 10
       height: 50,
       topIndent: 16,
       dotSize: 50,
+--186 lines: zoomButtonScaleFactor: 2-------------

   var ticks = blah.d;

   var getTimelineLabel = function (d, ticks) {
    var months = new Array('Jan','Feb','Mar','Apr'
    var index = Math.round((d.getTime() - selected
    var y = d.getFullYear();

    if (smallStep >= MILLIS_PER_YEAR ) {
        if (smallStep == 10 * MILLIS_PER_YEAR) ret
        if (smallStep == 5 * MILLIS_PER_YEAR)
            if (ticks.length > 25) return y%10==0
            else return y%5==0 ? y : '';
+-- 10 lines: ticks per year--------------------------
            else return '';
          else
            if (index%4 == 0) return months[d.getM
            else return '';
    }
    else if (smallStep >= MILLIS_PER_WEEK) {
        if (ticks.length < 20)
          if (index == ticks.length-1 || index ==
              return  months[d.getMonth()] + ' ' +
          else
            return months[d.getMonth()] + ' ' + d.
        else

          if (index%2 == 0)
            if (index >= ticks.length-2 || index =
              return  months[d.getMonth()] + ' ' +
            else
             return months[d.getMonth()] + ' ' + d
          else return '';
    }
    else if (smallStep >= MILLIS_PER_DAY) return d
    else return d.getDate();
   };

    /**
     * custom zoom behavior  when using the scroll
     */
    var myzoom = function (e, w) {
        speed = 1/7;// 1/48;
        var obj = e;
        width = w;
        function mousewheel() {
          var m = this.mouse();
```

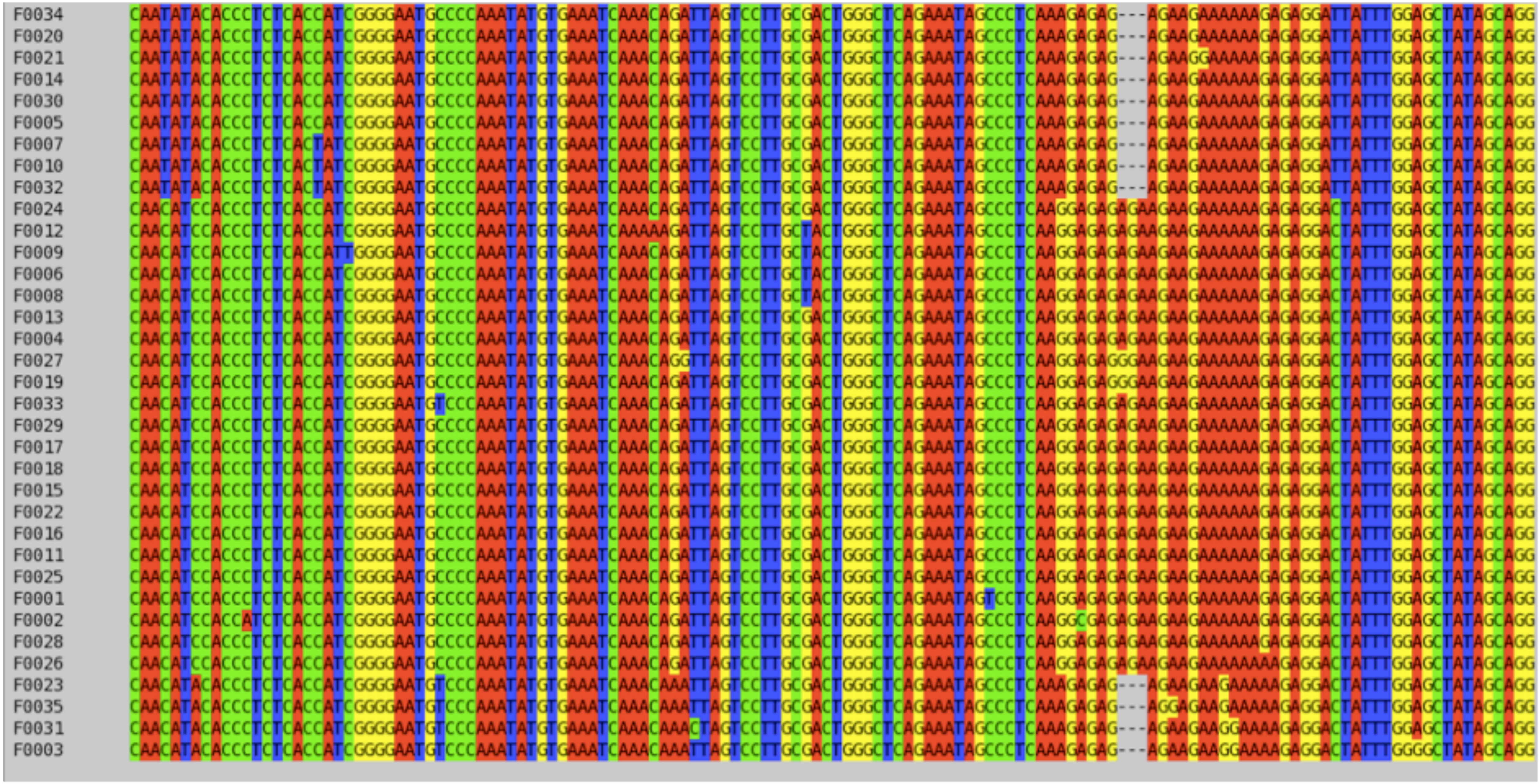
```
+-- 41 lines: Foreword here -------------------------
       labelColor: "white",
       labelFont: "14px sans-serif",
       backgroundColor: "black",
       dotBarStyle: 'green',//'rgba(255, 128, 128
       dotFillStyle: 'rgba(150,150,150,0.7)',
       dotStrokeStyle: "rgb(100, 100, 100)",
       dotHighlightFill: "rgb(255, 0, 0)",
       dotHighlightStroke: "rgba(100,100,100,0.3)
--------------------------------------------------
--------------------------------------------------
       dotBorderWidth: 1,
       dotBarPaddingLeft: 0, // 10
       dotBarPaddingRight: 0, // 10
       height: 50,
       topIndent: 16,
       dotSize: 50,
+--186 lines: zoomButtonScaleFactor: 2------------

   var ticks = blah.d;

   var getTimelineLabel = function (d, ticks) {
    var months = new Array('Jan','Feb','Mar','Apr
    var index = Math.round((d.getTime() - selecte
        var y = d.getFullYear();

    if (smallStep >= MILLIS_PER_YEAR ) {
        if (smallStep == 10 * MILLIS_PER_YEAR) re
        if (smallStep == 5 * MILLIS_PER_YEAR)
            if (ticks.length > 25) return y%10==0
            else return y%5==0 ? y : '';
+-- 10 lines: ticks per year-------------------------
            else return '';
          else
            if (index%4 == 0) return months[d.get
            else return '';
    }
    else if (smallStep >= MILLIS_PER_WEEK) {
        if (ticks.length < 20)
          return months[d.getMonth()] + ' ' + d.g
--------------------------------------------------
--------------------------------------------------
--------------------------------------------------
        else
          if (index%2 == 0) return months[d.getMo
--------------------------------------------------
--------------------------------------------------
--------------------------------------------------
--------------------------------------------------
--------------------------------------------------
          else return '';
    }
    else if (smallStep >= MILLIS_PER_DAY) return
    else return d.getDate();
   };

    /**
     * custom zoom behavior  when using the scrol
     */
    var myzoom = function (e, w) {
        speed = 1/48;
        var obj = e;
        width = w;
        function mousewheel() {
          var m = this.mouse();
```

```
+--- 41 lines: Foreword here ------------------------
       labelColor: "white",
       labelFont: "14px sans-serif",
       backgroundColor: "black",
       dotBarStyle: 'green',//'rgba(255, 128, 128,
       dotFillStyle: 'rgba(150,150,150,0.7)',
       dotStrokeStyle: "rgb(100, 100, 100)",
       dotHighlightFill: "rgb(255, 237, 160)",
       dotHighlightStroke: "rgba(100,100,100,0.3)"
       dotSelectFill: "coral",
       dotSelectStroke: "rgba(100,100,100,0.3)",
       dotBorderWidth: 1,
       dotBarPaddingLeft: 0, // 10
       dotBarPaddingRight: 0, // 10
       height: 50,
       topIndent: 16,
       dotSize: 50,
+--186 lines: zoomButtonScaleFactor: 2------------

   var ticks = blah.d;

   var getTimelineLabel = function (d, ticks) {
    var months = new Array('Jan','Feb','Mar','Apr'
    var index = Math.round((d.getTime() - selected
    var y = d.getFullYear();

    if (smallStep >= MILLIS_PER_YEAR ) {
        if (smallStep == 10 * MILLIS_PER_YEAR) ret
        if (smallStep == 5 * MILLIS_PER_YEAR)
            if (ticks.length > 25) return y%10==0
            else return y%5==0 ? y : '';
+--- 10 lines: ticks per year------------------------
            else return '';
          else
            if (index%4 == 0) return months[d.getM
            else return '';
    }
    else if (smallStep >= MILLIS_PER_WEEK) {
        if (ticks.length < 20)
          if (index == ticks.length-1 || index ==
              return  months[d.getMonth()] + ' ' +
          else
            return months[d.getMonth()] + ' ' + d.
        else

          if (index%2 == 0)
            if (index >= ticks.length-2 || index =
              return  months[d.getMonth()] + ' ' +
            else
             return months[d.getMonth()] + ' ' + d
          else return '';
    }
    else if (smallStep >= MILLIS_PER_DAY) return d
    else return d.getDate();
   };

    /**
     * custom zoom behavior  when using the scroll
     */
    var myzoom = function (e, w) {
        speed = 1/7;// 1/48;
        var obj = e;
        width = w;
        function mousewheel() {
          var m = this.mouse();
```

```
+--- 41 lines: Foreword here ------------------------
       labelColor: "white",
       labelFont: "14px sans-serif",
       backgroundColor: "black",
       dotBarStyle: 'green',//'rgba(255, 128, 128
       dotFillStyle: 'rgba(150,150,150,0.7)',
       dotStrokeStyle: "rgb(100, 100, 100)",
       dotHighlightFill: "rgb(255, 0, 0)",
       dotHighlightStroke: "rgba(100,100,100,0.3)
--------------------------------------------------
--------------------------------------------------
       dotBorderWidth: 1,
       dotBarPaddingLeft: 0, // 10
       dotBarPaddingRight: 0, // 10
       height: 50,
       topIndent: 16,
       dotSize: 50,
+--186 lines: zoomButtonScaleFactor: 2-----------

   var ticks = blah.d;

   var getTimelineLabel = function (d, ticks) {
    var months = new Array('Jan','Feb','Mar','Apr
    var index = Math.round((d.getTime() - selecte
        var y = d.getFullYear();

    if (smallStep >= MILLIS_PER_YEAR ) {
        if (smallStep == 10 * MILLIS_PER_YEAR) re
        if (smallStep == 5 * MILLIS_PER_YEAR)
            if (ticks.length > 25) return y%10==0
            else return y%5==0 ? y : '';
+--- 10 lines: ticks per year-----------------------
            else return '';
          else
            if (index%4 == 0) return months[d.get
            else return '';
    }
    else if (smallStep >= MILLIS_PER_WEEK) {
        if (ticks.length < 20)
          return months[d.getMonth()] + ' ' + d.g
--------------------------------------------------
--------------------------------------------------
--------------------------------------------------
        else
          if (index%2 == 0) return months[d.getMo
--------------------------------------------------
--------------------------------------------------
--------------------------------------------------
--------------------------------------------------
--------------------------------------------------
          else return '';
    }
    else if (smallStep >= MILLIS_PER_DAY) return
    else return d.getDate();
   };

    /**
     * custom zoom behavior  when using the scrol
     */
    var myzoom = function (e, w) {
        speed = 1/48;
        var obj = e;
        width = w;
        function mousewheel() {
          var m = this.mouse();
```

3rd source file here

Multiple Sequence Alignment (MSA)



```
F0034  CAATATACACCCTCTCACCATCGGGGAATGCCCCAAATATGTGAAATCAAACAGATTAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAAGAGAG---AGAAGAAAAAAGAGAGGATTATTTGGAGCTATAGCAGG
F0020  CAATATACACCCTCTCACCATCGGGGAATGCCCCAAATATGTGAAATCAAACAGATTAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAAGAGAG---AGAAGAAAAAAGAGAGGATTATTTGGAGCTATAGCAGG
F0021  CAATATACACCCTCTCACCATCGGGGAATGCCCCAAATATGTGAAATCAAACAGATTAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAAGAGAG---AGAAGGAAAAAGAGAGGATTATTTGGAGCTATAGCAGG
F0014  CAATATACACCCTCTCACCATCGGGGAATGCCCCAAATATGTGAAATCAAACAGATTAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAAGAGAG---AGAAGAAAAAAGAGAGGATTATTTGGAGCTATAGCAGG
F0030  CAATATACACCCTCTCACCATCGGGGAATGCCCCAAATATGTGAAATCAAACAGATTAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAAGAGAG---AGAAGAAAAAAGAGAGGATTATTTGGAGCTATAGCAGG
F0005  CAATATACACCCTCTCACCATCGGGGAATGCCCCAAATATGTGAAATCAAACAGATTAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAAGAGAG---AGAAGAAAAAAGAGAGGATTATTTGGAGCTATAGCAGG
F0007  CAATATACACCCTCTCACTATCGGGGAATGCCCCAAATATGTGAAATCAAACAGATTAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAAGAGAG---AGAAGAAAAAAGAGAGGATTATTTGGAGCTATAGCAGG
F0010  CAATATACACCCTCTCACTATCGGGGAATGCCCCAAATATGTGAAATCAAACAGATTAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAAGAGAG---AGAAGAAAAAAGAGAGGATTATTTGGAGCTATAGCAGG
F0032  CAATATACACCCTCTCACTATCGGGGAATGCCCCAAATATGTGAAATCAAACAGATTAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAAGAGAG---AGAAGAAAAAAGAGAGGATTATTTGGAGCTATAGCAGG
F0024  CAACATCCACCCTCTCACCATCGGGGAATGCCCCAAATATGTGAAATCAAACAGATTAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAGGAGAGAGAAGAAGAAAAAAGAGAGGACTATTTGGAGCTATAGCAGG
F0012  CAACATCCACCCTCTCACCATCGGGGAATGCCCCAAATATGTGAAATCAAAAAGATTAGTCCTTGCTACTGGGCTCAGAAATAGCCCTCAAGGAGAGAGAAGAAGAAAAAAGAGAGGACTATTTGGAGCTATAGCAGG
F0009  CAACATCCACCCTCTCACCATTGGGGAATGCCCCAAATATGTGAAATCAAACAGATTAGTCCTTGCTACTGGGCTCAGAAATAGCCCTCAAGGAGAGAGAAGAAGAAAAAAGAGAGGACTATTTGGAGCTATAGCAGG
F0006  CAACATCCACCCTCTCACCATCGGGGAATGCCCCAAATATGTGAAATCAAACAGATTAGTCCTTGCTACTGGGCTCAGAAATAGCCCTCAAGGAGAGAGAAGAAGAAAAAAGAGAGGACTATTTGGAGCTATAGCAGG
F0008  CAACATCCACCCTCTCACCATCGGGGAATGCCCCAAATATGTGAAATCAAACAGATTAGTCCTTGCTACTGGGCTCAGAAATAGCCCTCAAGGAGAGAGAAGAAGAAAAAAGAGAGGACTATTTGGAGCTATAGCAGG
F0013  CAACATCCACCCTCTCACCATCGGGGAATGCCCCAAATATGTGAAATCAAACAGATTAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAGGAGAGAGAAGAAGAAAAAAGAGAGGACTATTTGGAGCTATAGCAGG
F0004  CAACATCCACCCTCTCACCATCGGGGAATGCCCCAAATATGTGAAATCAAACAGATTAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAGGAGAGAGAAGAAGAAAAAAGAGAGGACTATTTGGAGCTATAGCAGG
F0027  CAACATCCACCCTCTCACCATCGGGGAATGCCCCAAATATGTGAAATCAAACAGGTTAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAGGAGAGGGAAGAAGAAAAAAGAGAGGACTATTTGGAGCTATAGCAGG
F0019  CAACATCCACCCTCTCACCATCGGGGAATGCCCCAAATATGTGAAATCAAACAGATTAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAGGAGAGGGAAGAAGAAAAAAGAGAGGACTATTTGGAGCTATAGCAGG
F0033  CAACATCCACCCTCTCACCATCGGGGAATGTCCCAAATATGTGAAATCAAACAGATTAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAGGAGAGAGAAGAAGAAAAAAGAGAGGACTATTTGGAGCTATAGCAGG
F0029  CAACATCCACCCTCTCACCATCGGGGAATGCCCCAAATATGTGAAATCAAACAGATTAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAGGAGAGAGAAGAAGAAAAAAGAGAGGACTATTTGGAGCTATAGCAGG
F0017  CAACATCCACCCTCTCACCATCGGGGAATGCCCCAAATATGTGAAATCAAACAGATTAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAGGAGAGAGAAGAAGAAAAAAGAGAGGACTATTTGGAGCTATAGCAGG
F0018  CAACATCCACCCTCTCACCATCGGGGAATGCCCCAAATATGTGAAATCAAACAGATTAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAGGAGAGAGAAGAAGAAAAAAGAGAGGACTATTTGGAGCTATAGCAGG
F0015  CAACATCCACCCTCTCACCATCGGGGAATGCCCCAAATATGTGAAATCAAACAGATTAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAGGAGAGAGAAGAAGAAAAAAGAGAGGACTATTTGGAGCTATAGCAGG
F0022  CAACATCCACCCTCTCACCATCGGGGAATGCCCCAAATATGTGAAATCAAACAGATTAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAGGAGAGAGAAGAAGAAAAAAGAGAGGACTATTTGGAGCTATAGCAGG
F0016  CAACATCCACCCTCTCACCATCGGGGAATGCCCCAAATATGTGAAATCAAACAGATTAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAGGAGAGAGAAGAAGAAAAAAGAGAGGACTATTTGGAGCTATAGCAGG
F0011  CAACATCCACCCTCTCACCATCGGGGAATGCCCCAAATATGTGAAATCAAACAGATTAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAGGAGAGAGAAGAAGAAAAAAGAGAGGACTATTTGGAGCTATAGCAGG
F0025  CAACATCCACCCTCTCACCATCGGGGAATGCCCCAAATATGTGAAATCAAACAGATTAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAGGAGAGAGAAGAAGAAAAAAGAGAGGACTATTTGGAGCTATAGCAGG
F0001  CAACATCCACCCTCTCACCATCGGGGAATGCCCCAAATATGTGAAATCAAACAGATTAGTCCTTGCGACTGGGCTCAGAAATAGTCCTCAAGGAGAGAGAAGAAGAAAAAAGAGAGGACTATTTGGAGCTATAGCAGG
F0002  CAACATCCACCATCTCACCATCGGGGAATGCCCCAAATATGTGAAATCAAACAGATTAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAGGCGAGAGAAGAAGAAAAAAGAGAGGACTATTTGGAGCTATAGCAGG
F0028  CAACATCCACCCTCTCACCATCGGGGAATGCCCCAAATATGTGAAATCAAACAGATTAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAGGAGAGAGAAGAAGAAAAAAGAGAGGACTATTTGGAGCTATAGCAGG
F0026  CAACATCCACCCTCTCACCATCGGGGAATGCCCCAAATATGTGAAATCAAACAGATTAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAGGAGAGAGAAGAAGAAAAAAAAGAGGACTATTTGGAGCTATAGCAGG
F0023  CAACATACACCCTCTCACCATCGGGGAATGTCCCAAATATGTGAAATCAAACAAATTAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAAGAGAG---AGAAGAAGAAAAAGAGGACTATTTGGAGCTATAGCAGG
F0035  CAACATACACCCTCTCACCATCGGGGAATGTCCCAAATATGTGAAATCAAACAAATTAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAAGAGAG---AGGAGAAGAAAAAGAGGACTATTTGGAGCTATAGCAGG
F0031  CAACATACACCCTCTCACCATCGGGGAATGTCCCAAATATGTGAAATCAAACAAACTAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAAGAGAG---AGAAGAAGGAAAAGAGGACTATTTGGAGCTATAGCAGG
F0003  CAACATACACCCTCTCACCATCGGGGAATGTCCCAAATATGTGAAATCAAACAAATTAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAAGAGAG---AGAAGAAGGAAAAGAGGACTATTTGGGGCTATAGCAGG
```

Multiple sequence alignment: find more subtle patterns & find common patterns between all sequence.

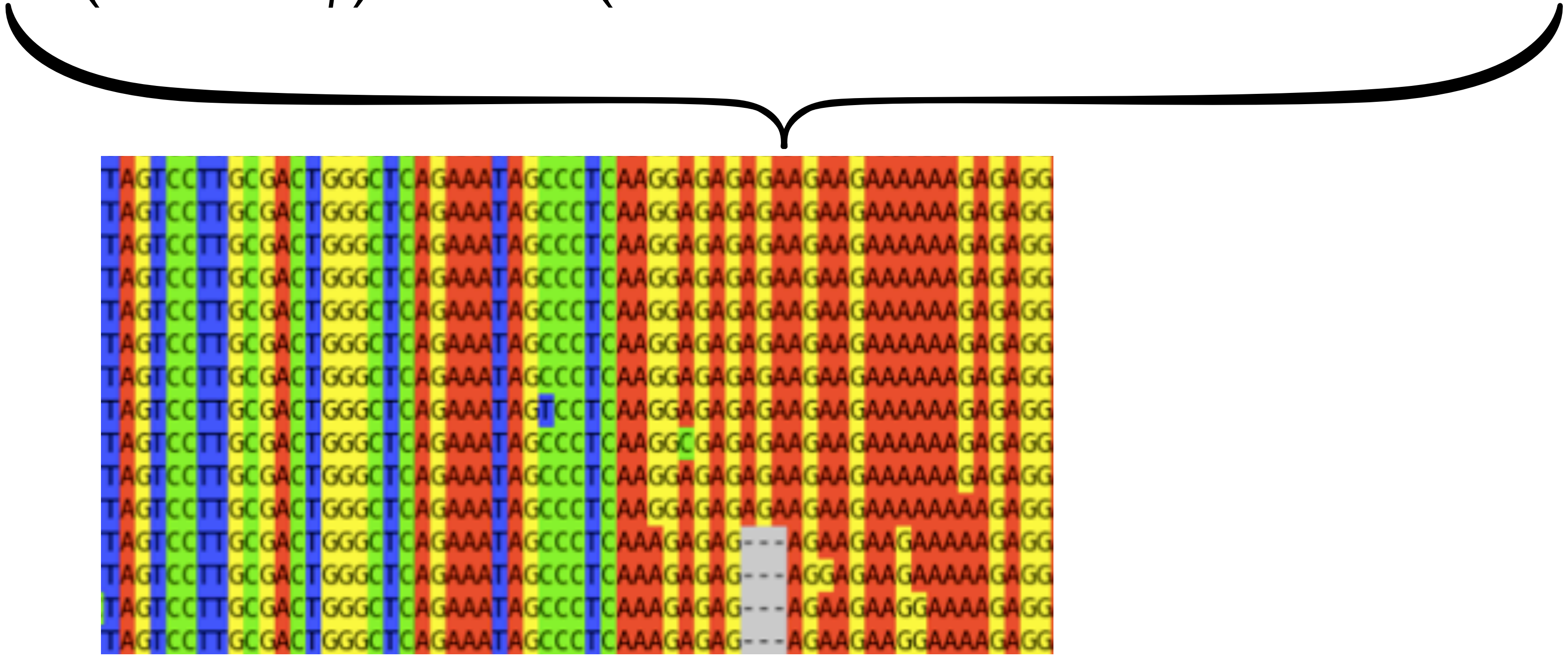
Generalizing Alignment to > 2 Strings

Input: Sequences $S_1, S_2, \ldots, S_p$

Let $cost(x_1,x_2,\ldots x_p)$ be a user-supplied function that computes the quality of a column: an alignment between characters $x_1, x_2, \ldots x_p$.

- **Goal**: find alignment M to **minimize** $\sum$ cost of the columns:

$cost(x_1,x_2,\ldots x_p)$ = cost()

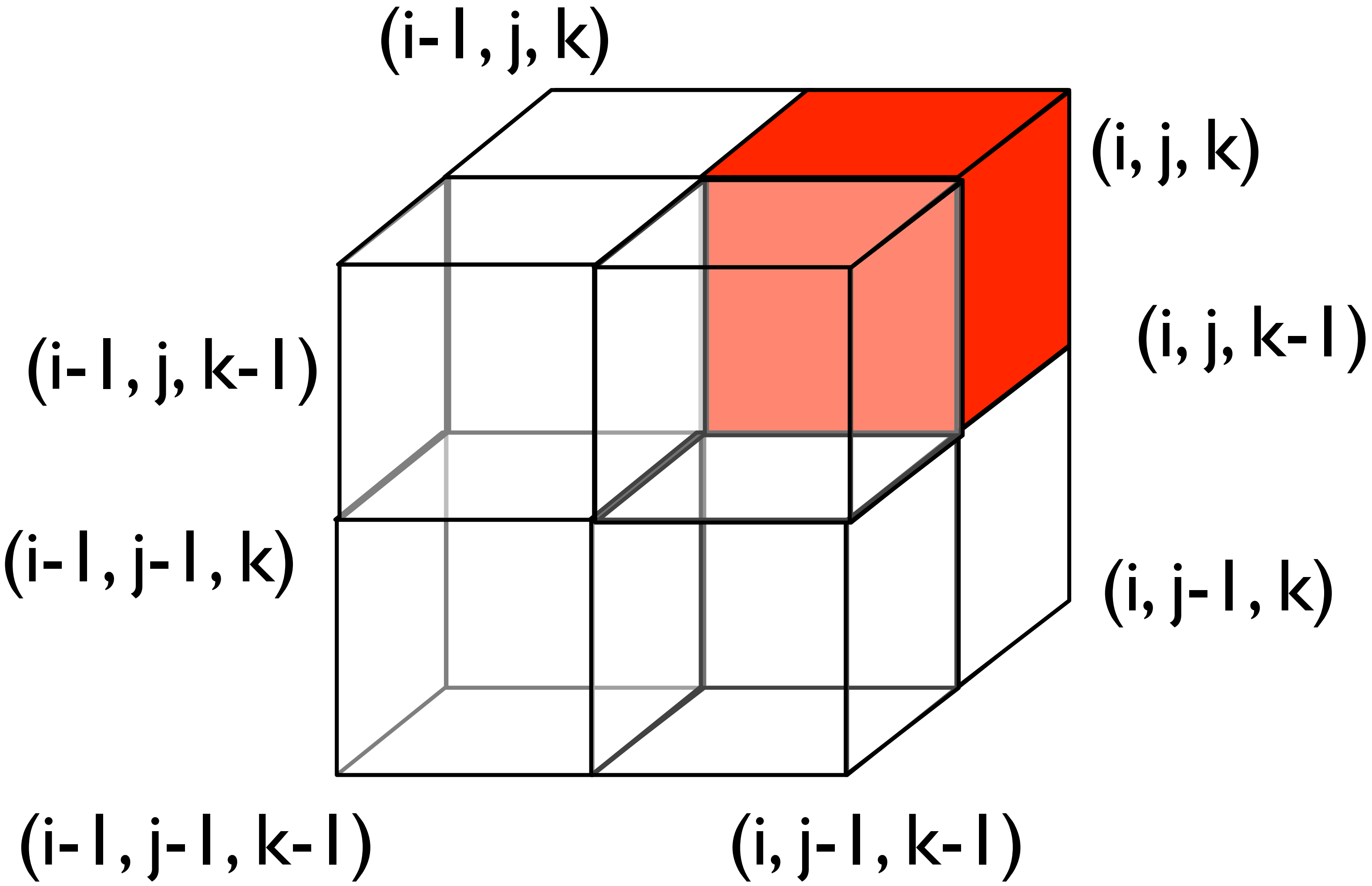


```
TAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAGGAGAGAGAAGAAGAAAAAAGAGAGG
TAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAGGAGAGAGAAGAAGAAAAAAGAGAGG
TAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAGGAGAGAGAAGAAGAAAAAAGAGAGG
TAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAGGAGAGAGAAGAAGAAAAAAGAGAGG
TAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAGGAGAGAGAAGAAGAAAAAAGAGAGG
TAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAGGAGAGAGAAGAAGAAAAAAGAGAGG
TAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAGGAGAGAGAAGAAGAAAAAAGAGAGG
TAGTCCTTGCGACTGGGCTCAGAAATAGTCCTCAAGGAGAGAGAAGAAGAAAAAAGAGAGG
TAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAGGCGAGAGAAGAAGAAAAAAGAGAGG
TAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAGGAGAGAGAAGAAGAAAAAAGAGAGG
TAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAGGAGAGAGAAGAAGAAAAAAAAGAGG
TAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAAGAGAG---AGAAGAAGAAAAAGAGG
TAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAAGAGAG---AGGAGAAGAAAAAGAGG
TAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAAGAGAG---AGAAGAAGGAAAAGAGG
TAGTCCTTGCGACTGGGCTCAGAAATAGCCCTCAAAGAGAG---AGAAGAAGGAAAAGAGG
```

Slow Dynamic Programming

Suppose you had just 3 sequences.

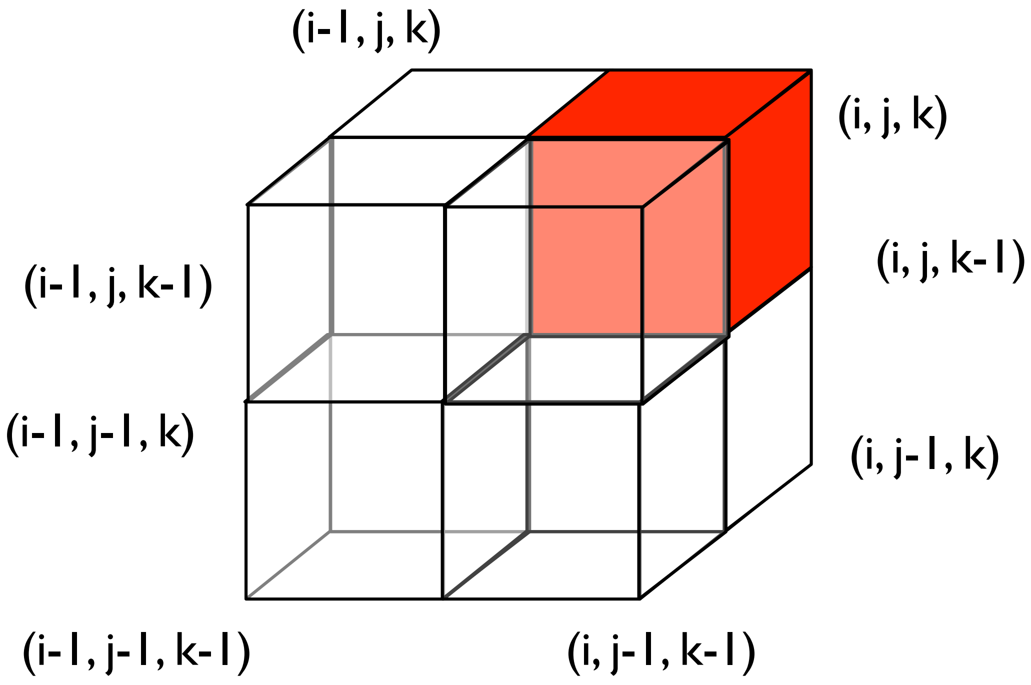
Apply the same DP idea as sequence alignment for 2 sequences, but now with a 3-dimensional matrix



DP Recurrence for 3 sequences

$$A[i,j,k] = \min \begin{cases} \mathrm{cost}(x_i, y_j, z_k) + A[i-1, j-1, k-1] \\ \mathrm{cost}(x_i, -, -) + A[i-1, j, k] \\ \mathrm{cost}(x_i, y_j, -) + A[i-1, j-1, k] \\ \mathrm{cost}(-, y_j, z_k) + A[i, j-1, k-1] \\ \mathrm{cost}(-, y_j, -) + A[i, j-1, k] \\ \mathrm{cost}(x_i, -, z_k) + A[i-1, j, k-1] \\ \mathrm{cost}(-, -, z_k) + A[i, j, k-1] \end{cases}$$

Every possible pattern for the gaps.



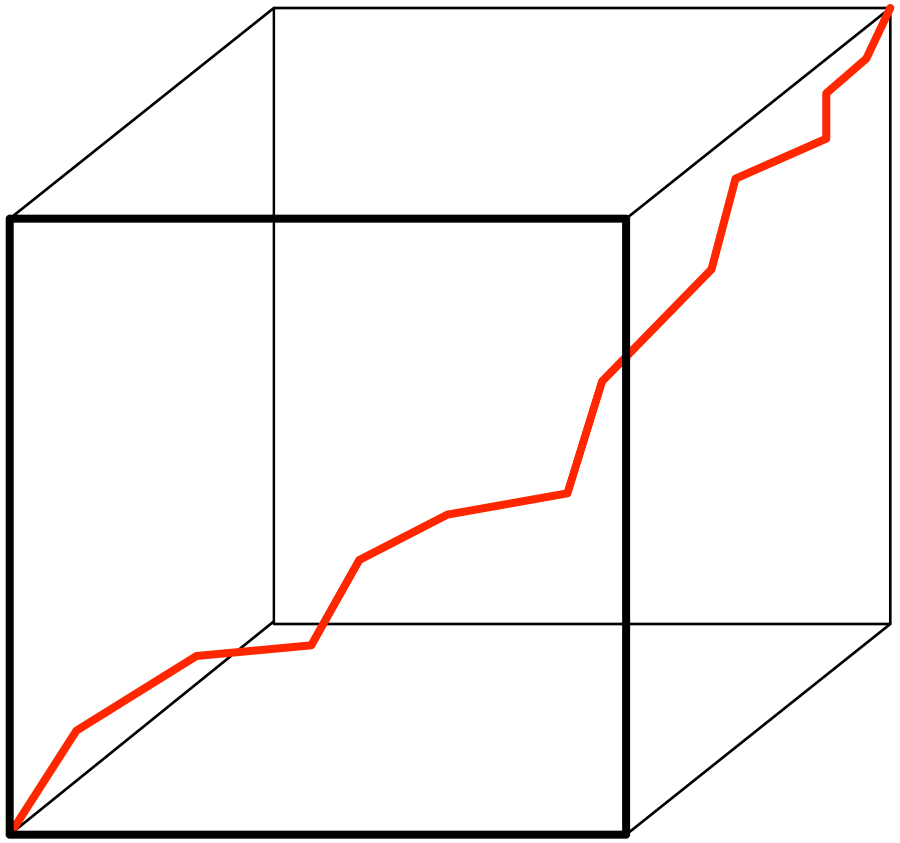
Running time

- n^3 subproblems, each takes 2^3 time $\Rightarrow O(n^3)$ time.

- For p sequences: n^p subproblems, each takes 2^p time for the max and p^2 to compute cost() $\Rightarrow O(p^2 n^p 2^p)$

- Even $O(n^3)$ is often too slow for the length of sequences encountered in practice.

- One solution: approximation algorithm.



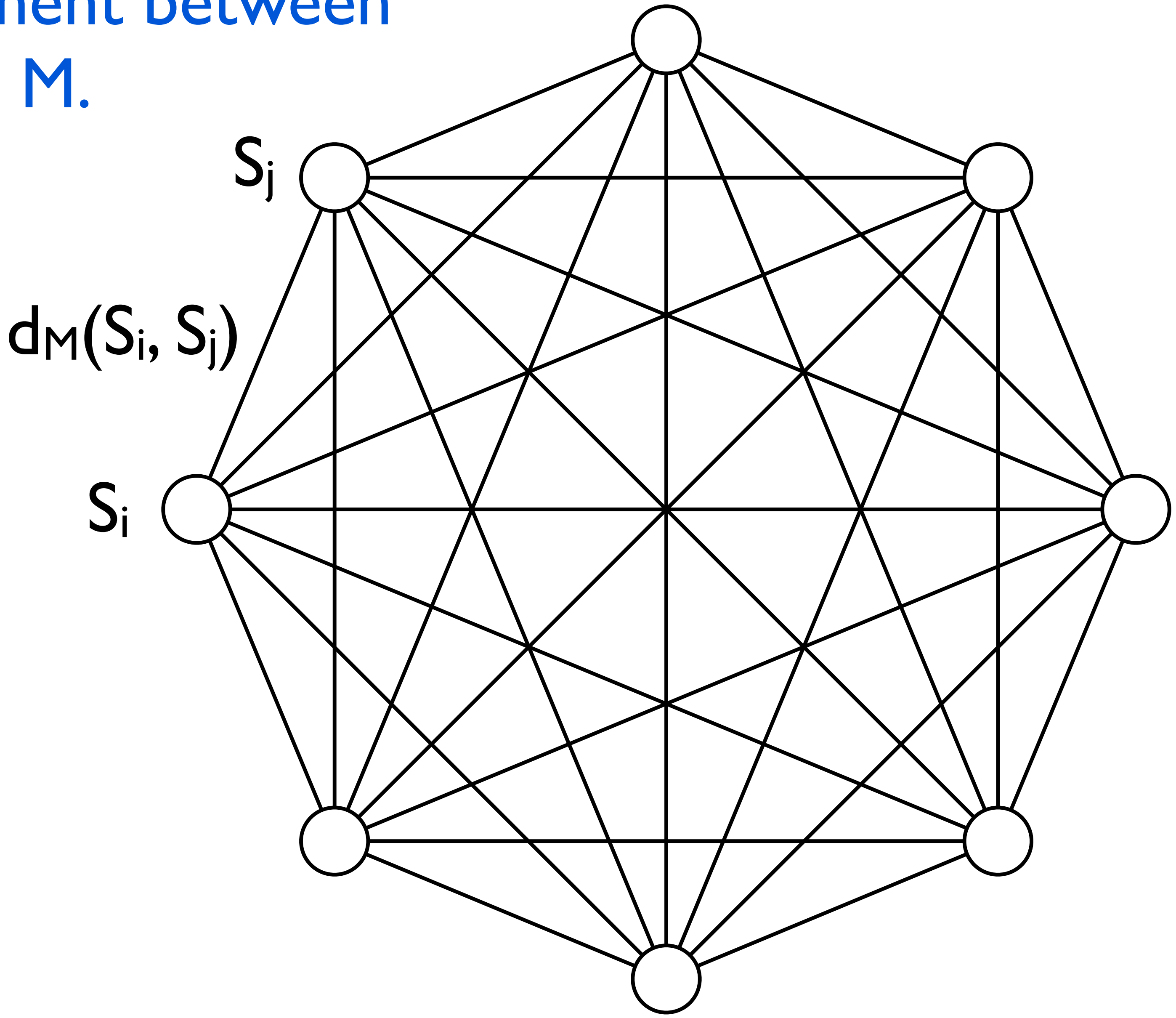
SP-Score

A particular cost() function, the SP-Score, is commonly used and allows us to design an approximation algorithm for the MSA problem.

$d_M(S_i, S_j)$ = the cost of the alignment between S_i and S_j **as implied by MSA** M.

SP-Score(M) = $\sum_{i<j} d_M(S_i, S_j)$

= sum of all the scores of the pairwise alignments implied by M.



MSA

- A multiple sequence alignment (MSA) implies a pairwise alignment between every pair of sequences.
- This implied alignment need not be optimal, however:

match = -1, a mismatch = 1, gap = 2

Sequences: AT, A, T, AT, AT

Optimal MSA:

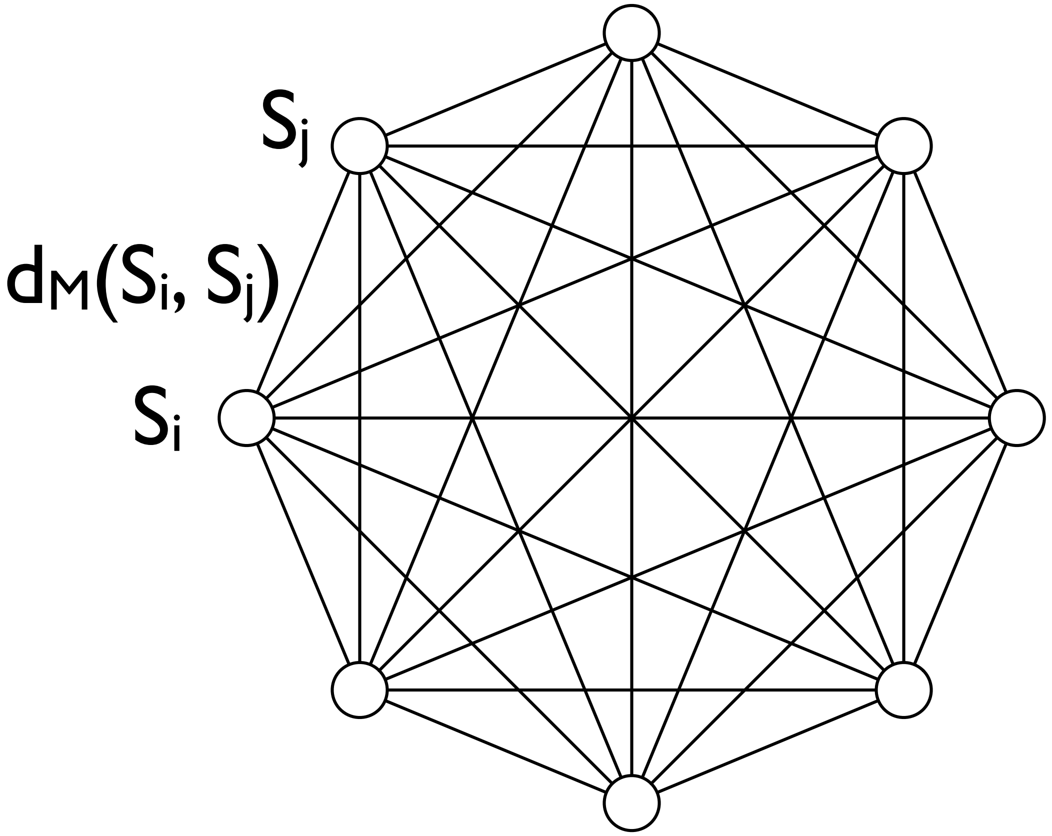
```
  AT
  A-
  -T
  AT
  AT
+2 +2 = 4
```

Optimal Alignment between A and T:

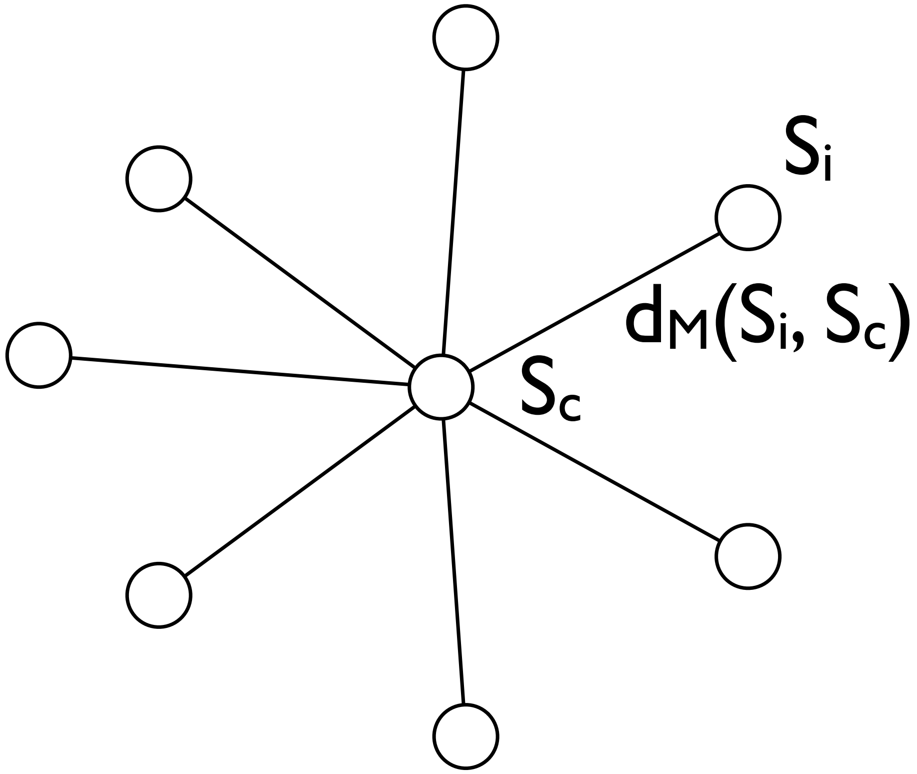
```
A
T
+1
```

(A,A), (A,-), (A,A), (A,A), (A, -), (A,A), (A,A) (-,A), (-,A), (A,A)
-1 + 2 -1 -1 +2 -1 -1 +2 +2 -1 = +2

Star Alignment Approximation



SP-Score



Star-Score =

$\sum_i d_M(S_i, S_c)$

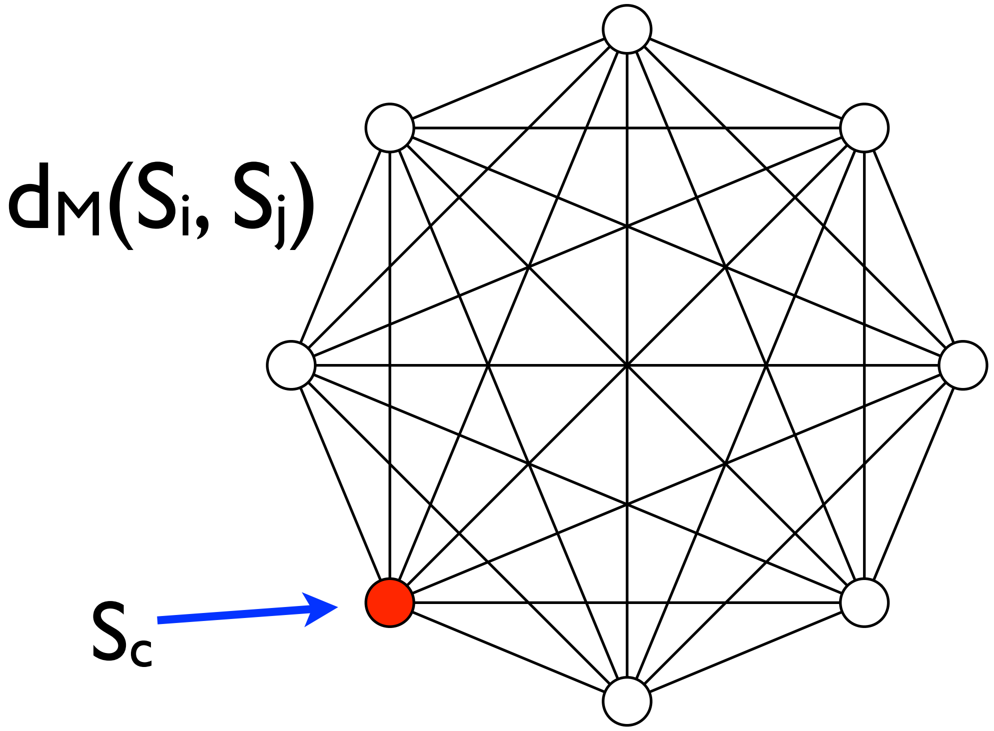
Star Alignment Algorithm

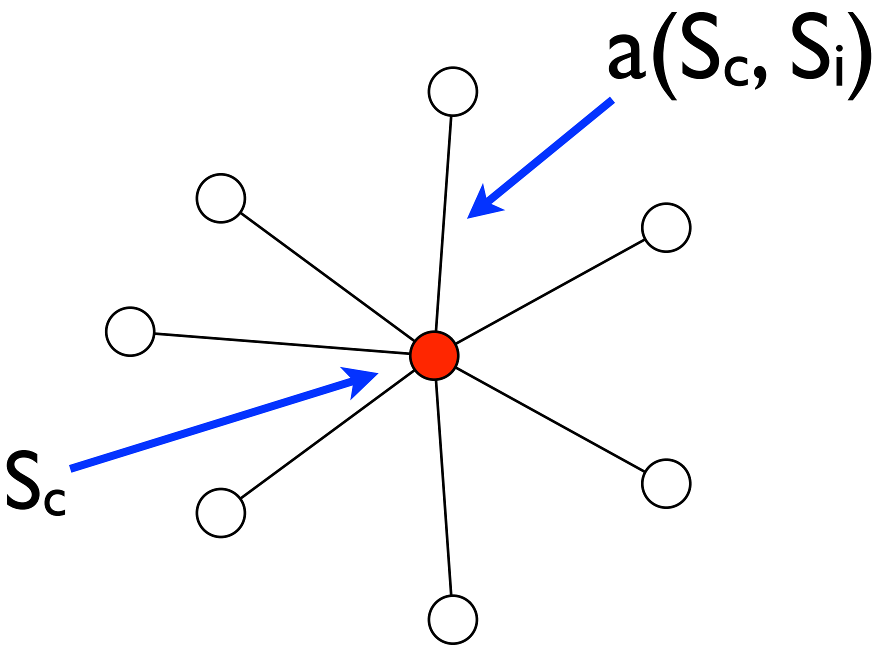
Input: sequences $S_1, S_2, \ldots, S_p$

- Build all $O(p^2)$ pairwise alignments.
- Let S_c = the sequence in $S_1, S_2, \ldots, S_p$ that is closest to the others. That is, choose S_c to minimize:

$$\sum_{i \neq c} a(S_c, S_i)$$

- *Progressively align* all other sequences to S_c.





Progressive Alignment

- Build a multiple sequence alignment up from pairwise alignments.

Start with an alignment between S_c and some other sequence:

```
SC YFPHFDLSHGSAQVKAHGKKVGDALTLAVGHLDDLPGAL
S1 YFPHFDLSHG-AQVKG--KKVADALTNAVAHVDDMPNAL
```

Add 3rd sequence, say S2, and use the SC - S2 alignment as a guide, adding spaces into the MSA as needed.

SC - S2 alignment:

```
SC YFPHF-DLS-----HGSAQVKAHGKKVGDALTLAVGHL----DDLPGAL
S2 FFPKFKGLTTADQLKKSADVRWHAERII----NAVNDAVASMDDTEKMS
```

New {SC, S1, S2} alignment (red gaps added in S1):

```
SC YFPHF-DLS-----HGSAQVKAHGKKVGDALTLAVGHL----DDLPGAL
S1 YFPHF-DLS-----HG-AQVKG--KKVADALTNAVAHV----DDMPNAL
S2 FFPKFKGLTTADQLKKSADVRWHAERII----NAVNDAVASMDDTEKMS
```

Continue with S3, S4, ...

Performance

Assume the cost function satisfies the triangle inequality:

$$\text{cost}(x,y) \le \text{cost}(x, z) + \text{cost}(z,y)$$

Example: $\underbrace{\text{cost}(A, C)}_{\substack{\text{cost of 1} \\ \text{mutation from} \\ A \rightarrow C}} \le \underbrace{\text{cost}(A,T) + \text{cost}(T,C)}_{\substack{\text{cost of a mutation} \\ \text{from } A \rightarrow T \text{ and} \\ \text{then from } T \rightarrow C}}$

STAR = cost of result of star algorithm under SP-score

OPT = cost of optimal multiple sequence alignment (under SP-score)

Theorem. If cost satisfies the triangle inequality, then STAR $\le$ 2×OPT.

Example: if optimal alignment has cost 10, the star alignment will have cost $\le$ 20.

Proof (1)

Theorem. If cost satisfies the triangle inequality, then STAR ≤ 2OPT.

$$\frac{\text{STAR}}{\text{OPT}} \le 2$$

For some *B* we will
prove the 2 statements:

$$\boxed{\begin{array}{l}\text{STAR} \le 2B \\ \text{OPT} \ge B\end{array}}$$

This will imply:

$$\Longrightarrow \frac{\text{STAR}}{\text{OPT}} \le \frac{2B}{B} = 2$$

Proof (2)

Theorem. If cost satisfies the triangle inequality, then STAR ≤ 2OPT.

$$2 \cdot \mathrm{STAR} = \sum_{ij} d_{\mathrm{STAR}}(S_i, S_j)$$ defn of SP-score

by triangle inequality $$\leq \sum_{ij} (d_{\mathrm{STAR}}(S_i, S_c) + d_{\mathrm{STAR}}(S_c, S_j))$$

because STAR alignment is optimal for pairs involving Sc $$= \sum_{ij} (\mathsf{a}(S_i, S_c) + \mathsf{a}(S_c, S_j))$$

distribute ∑ $$= \sum_{ij} \mathsf{a}(S_i, S_c) + \sum_{ij} \mathsf{a}(S_c, S_j)$$

$$\leq 2p \sum_{i} \mathsf{a}(S_i, S_c)$$

sums are the same and each term appears ≤ p (# of sequences) times.

Proof (3)

Theorem. If cost satisfies the triangle inequality, then STAR ≤ 2OPT.

$$2 \cdot \mathrm{OPT} = \sum_{ij} d_{\mathrm{OPT}}(S_i, S_j)$$

defn of SP-score

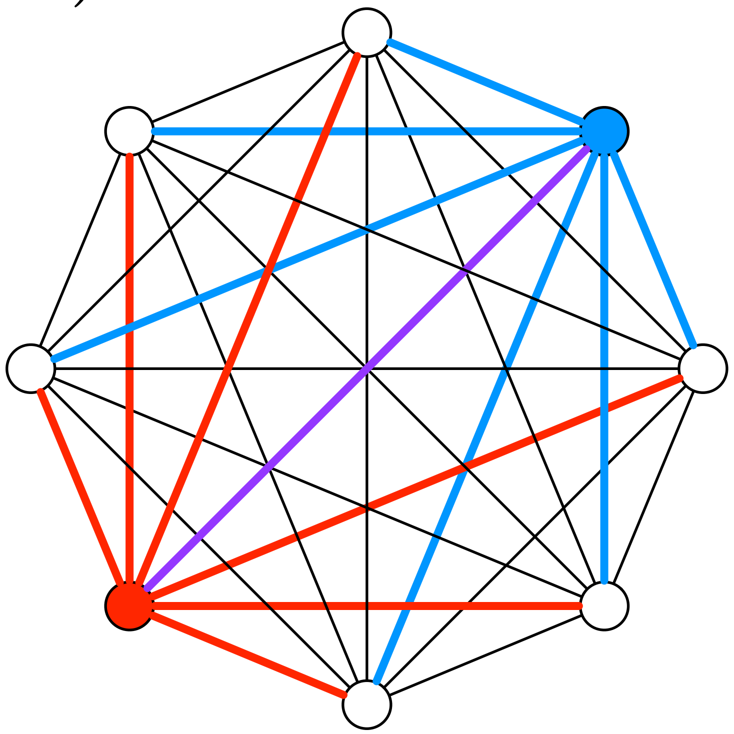
optimal pairwise alignment is ≤ pairwise alignment induced by any MSA

$$\geq \sum_{ij} \mathsf{a}(S_i, S_j)$$

$$\geq p \sum_{i} \mathsf{a}(S_i, S_c)$$

sum of cost of all pairwise alignments is = the sum of *p* different stars.

We chose S_c because it was the lowest-cost star.



End of Proof

For some *B* we will prove the 2 statements:

$$\boxed{\begin{array}{l}\text{STAR} \leq 2B \\ \text{OPT} \geq B\end{array}}$$

This will imply:

$$\Longrightarrow \frac{\text{STAR}}{\text{OPT}} \leq \frac{2B}{B} = 2$$

$$\boxed{\begin{array}{rcl} 2 \cdot STAR & \leq & 2p \sum_i \mathsf{a}(S_i, S_c) \\ 2 \cdot OPT & \geq & p \sum_i \mathsf{a}(S_i, S_c) \end{array}}$$

$$\Longrightarrow \frac{\text{STAR}}{\text{OPT}} \leq \frac{2p \sum_i \mathsf{a}(S_i, S_c)}{p \sum_i \mathsf{a}(S_i, S_c)} = 2$$

Consensus Sequence

For every column j,
choose $c \in \Sigma$ that
minimizes $\sum_i \mathrm{cost}(c, S_i[j])$

(typically this means the
most common letter)

```
S1 YFPHF-DLS-----HGSAQVKAHGKKVG-----DALTLAVAHLDDLPGAL
S2 YFPHF-DLS-----HG-AQVKG—GKKVA-----DALTNAVAHVDDMPNAL
S3 FFPKFKGLTTADQLKKSADVRWHAERII-----NAVNDAVASMDDTEKMS
S4 LFSFLKGTSEVP--QNNPELQAHAGKVFKLVYEAAIQLQVTGVVVTDATL
CO YFPHFKDLS-----HGSAQVKAHGKKVG-----DALTLAVAHVDDTPGAL
```

- Consensus is a summarization of the whole alignment.
- Consensus sequence is sometimes used as an estimate for the ancestral sequence.
- Sometimes the MSA problem is formulated as: find MSA M that minimizes: $\sum_i d_M(CO, S_i)$

Profiles

- Another way to summarize an MSA:

```
S1 ACG-TT-GA
S2 ATC-GTCGA
S3 ACGCGA-CC
S4 ACGCGT-TA
```

Column in the alignment

Character

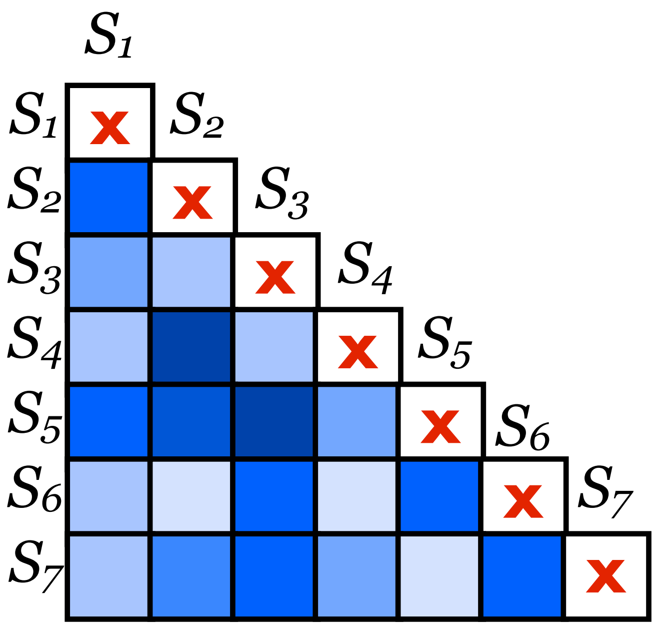
	1	2	3	4	5	6	7	8	9
A	1	0	0	0	0	0.25	0	0	0.75
C	0	0.75	0.25	0.5	0	0	0.25	0.25	0.25
G	0	0	0.75	0	0.75	0	0	0.5	0
T	0	0.25	0	0	0.25	0.75	0	0.25	0
-	0	0	0	0.5	0	0	0.75	0	0

Call this profile matrix R

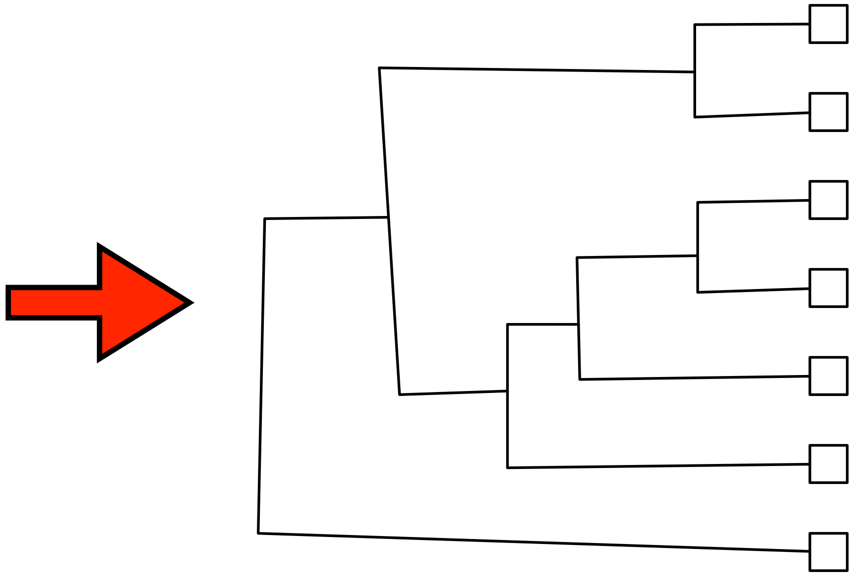
Fraction of time given column had the given character

CLUSTLW

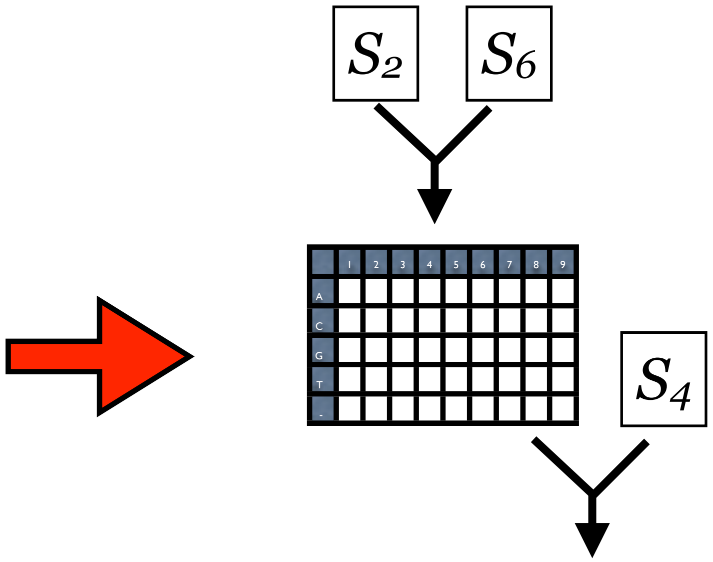
- CLUSTLW is a widely used, “classical” heuristic multiple aligner.
- Not the fastest, not the most accurate, but pretty good.
- Large # of heuristic tricks included in the software, but basic idea is straightforward:



Step (1): Build pairwise distance matrix



Step (2): Build guide tree



Step (3): Align sequences / **sets of sequences** from the most similar to least similar

Profile-based Alignment

gap in profile
introduced to
better fit sequence

R =

	1	2	3	4	5	6	7	8	9
A	1	0	0	0	0	0.25	0	0	0.75
C	0	0.75	0.25	0.5	0	0	0.25	0.25	0.25
G	0	0	0.75	0	0.75	0	0	0.5	0
T	0	0.25	0	0	0.25	0.75	0	0.25	0
-	0	0	0	0.5	0	0	0.75	0	0

A C C - A G A C G A

Score of matching character x with column j of the profile:

$$P(x,j) = \sum_{c\in\Sigma} \mathrm{sim}(x,c) \times R[c,j]$$

sim(x,c) = how similar character x is to character c.

$$A[i,j] = \max \begin{cases} A[i-1,j-1] + P(x_i,j) & \text{align } x_i \text{ to column } j \\ A[i-1,j] + \text{gap} & \text{introduce gap into profile} \\ A[i,j-1] + P(\text{``\_``},j) & \text{introduce gap into } x \end{cases}$$

Recap

- Multiple sequence alignments (MSAs) are a fundamental tool. They help reveal subtle patterns, compute consistent distances between sequences, etc.
- Quality of MSAs often measured using the SP-score: sum of the scores of the pairwise alignments implied by the MSA.
- Same DP idea as pairwise alignment leads to exponentially slow algorithm for MSA for general p.
- 2-approximation obtainable via star alignments.
- MSAs often used to create profiles summarizing a family of sequences.