Sequence Alignment using *Align*

Part I: Introduction

Richard M. Salter
Oberlin College

Supported by NSF CCLI DUE 0618252

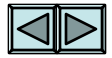
Sequence alignment

- According to Krane & Raymer, *Fundamental Concepts of Bioinformatics*:

> *In a very real sense, any alignment between two or more nucleotide or amino acid sequences represents an explicit hypothesis regarding the evolutionary history of those sequences. As a direct result, comparisons of related protein and nucleotide sequences have facilitated many recent advances in understanding the information content and function of genetic sequences. For this reason, techniques for aligning and comparing sequences, and for searching sequence databases for similar sequences, have become cornerstones of bioinformatics.*

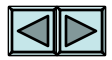
- In this module you will learn about the algorithms used in pairwise sequence alignment.

- You will use the *Align* tool to help you understand how these algorithms work, and to experiment with sequence matching under various constraints.



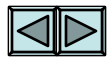
Align

- *Align* is a Java application that presents 3 essential alignment algorithms. *Align* presents these algorithms with the following goals in mind:
  - Using Auto mode, *Align* demonstrates how the algorithms determine the best match.
  - Using Manual mode, *Align* allows you, the student, to attempt your own match, and compares that result with the result produced by the algorithms.



Prerequisites and Synopsis

- This module will work well with a text that fully describes sequence alignment.
  - an excellent text describing sequence alignment is Krane & Raymer, *Fundamental Concepts of Bioinformatics.*
  - Additional on-line references are provided in the *Instructor's Guide.*
- Part I introduces sequence alignment.
- Part II describes the basic dynamic programming algorithm.
- Part III discusses the scoring schemes used in sequence alignment.
- Part IV considers semi-global and local variations of the basic algorithm.



Sequences

- By **sequence** we mean a sequence of symbols representing either nucleotides or amino acids. Letter sequences are often called **strings**.

- In the simplest cases specific nucleotides and amino acids are represented by specific letters.
  - The 5 nucleotides (adenine, cytosine, thymine, guanine and uracil) are represented respectively by `a, c, t, g` and `u`.
  - The 20 amino acids are similarly represented with 20 alphabetic symbols.
  - Examples:

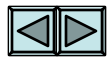
    DNA: `acgggcattacgtgccctgg`

    RNA: `acugcauggcgguuagcu`

    Protein: `GGDFHIKLVDMNPQQLKKLAC`

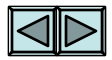
- In the previous examples, each letter stands for a specific nucleotide or amino acid. In the symbol systems used in bioinformatics, certain letters may represent several possible entities. For example, when matching nucleotides, `m` can represent either adenine or cytosine (`a` or `c`), while `k` can represent either thymine or guanine (`t` or `g`).

- Complete descriptions of the symbols used in bioinformatics will be given later.



Sequence Alignment

- An alignment between 2 sequences is a pairwise match between the characters of each sequence.
- Since sequence pairs rarely match 100%, an alignment must be able to express the degree of similarity between sequence pairs.
- Evolution can introduce 3 types of changes at any given point in a sequence:
  - A *mutation* that replaces one character with another;
  - An *insertion* that adds one or more positions; or
  - A *deletion* that eliminates one or more positions.
- In order to account for insertions and deletions, sequence patterns include *gaps* representing unknown inserted or deleted subsequences.



Examples

- Consider sequences:

  `aatctata`

  `aagata`

- Which of these matches shown do you like best?

- Alignments without gaps
  - `aatctata`
    `aagata`

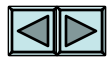
  - `aatctata`
    ` aagata`

  - `aatctata`
    `  aagata`

- Alignments with gaps
  - `aatctata`
    `aag-at-a`

  - `aatctata`
    `aa-g-ata`

  - `aatctata`
    `aa--gata`



Scoring

- Given a possible match, we need to assign a numerical score that reflects the degree of similarity between the 2 sequences.

- For a gapped alignment, a *simple score* can be based on the following parameters summed over each pair of corresponding letters *x* and *y*:

  - *Match score*:
    - if neither *x* nor *y* is a gap, and $x = y$;
  - *Mismatch score:*
    - if neither *x* nor *y* is a gap, and $x \neq y$;
  - *Gap penalty:*
    - if *x* or *y* or both are gaps.

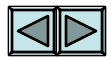
- Example: Let's assign the match score to 1, the mismatch score to 0, and the gap penalty to -1.

- Consider the match

```
aatctata
aa-g-ata
```

- Scoring this match
  - Column 1: `a a` match: 1
  - Column 2: `a a` match: 1
  - Column 3: `t -` gap: -1
  - Column 4: `c g` mismatch: 0
  - Column 5: `t -` gap: -1
  - Column 6: `a a` match: 1
  - Column 7: `t t` match: 1
  - Column 8: `a a` match: 1
  -

  - Total score 3



Exercise 1.1: Scoring By Hand

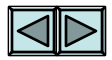
- Now try to score the examples we saw before using the scoring parameters used on the previous page:
  - Match: 1
  - Mismatch: 0
  - Gap Penalty: -1

  (for short, we'll call this [1,0,-1])
- Click the button to view the solutions
- Did your most favored match score highest?

- Exercise 1

```
a) aatctata
   aagata

b) aatctata
    aagata

c) aatctata
     aagata

d) aatctata
   aag-at-a

e) aatctata
   aa-g-ata

f) aatctata
   aa--gata
```

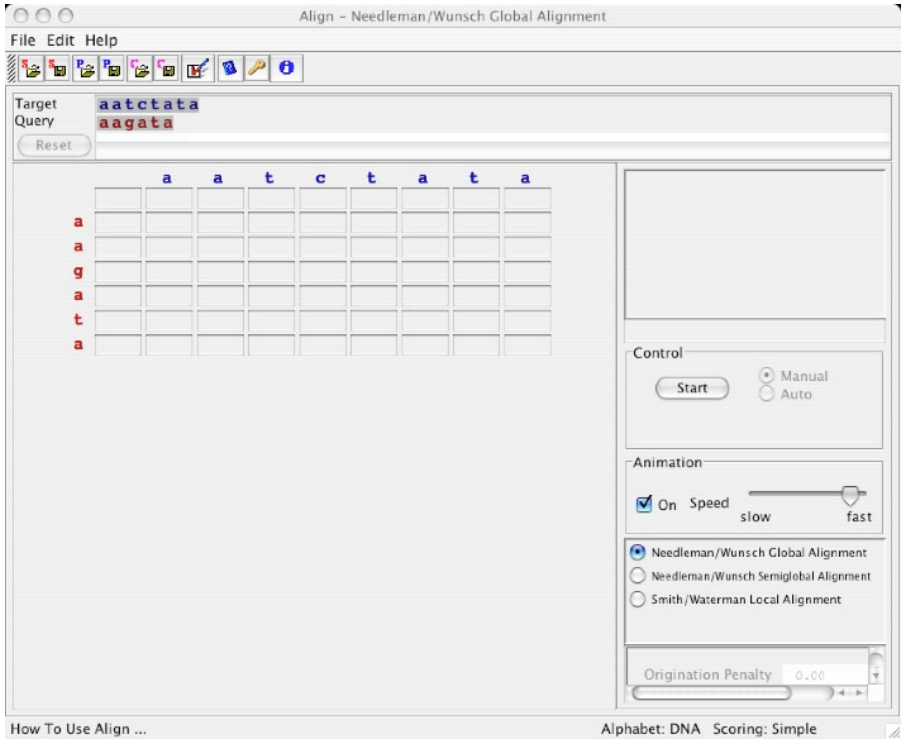
Exercise 1.1 Solution



Using *Align*

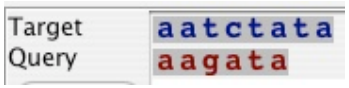
- We're ready to use *Align* to score the alignment.
  If you haven't already done so, install the *Align* software as described in the "readme" file that came with the package.

  Launch *Align;* you should see the screen pictured below.

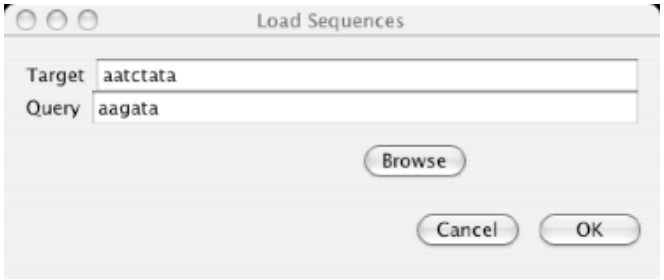


Look at the sequences

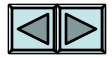
- Note that the sequence pair shown in the program are the pair we have been working with.



- If not, please do the following:
  - Select **File | Load Sequences …** (i.e. on the **File** menu select **Load Sequences …**).
  - When the Load Sequences dialog appears, enter the sequences into the target and query windows.
  - Click OK



  - When you exit *Align* be sure to use **File | Save**. That way the sequences you entered will be saved and should appear the next time you run *Align.*



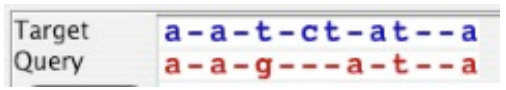
Using *Align* by hand

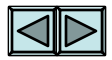
- You are now ready to try your hand at aligning the sequence pair.

- Look again at the sequence pair at the top of the Align window.
  - Click the Start button in the Control panel.
  - Using your mouse, point to the red "g" in the query string.
  - Right-click (or shift-click for single-button mouse users) and note that a gap appears. Left click on this gap to delete it.
  - Try entering and deleting gaps at various places in the string.

- Using the mouse, you can place gaps anywhere you want in either string.

- See if you can create the following pattern:





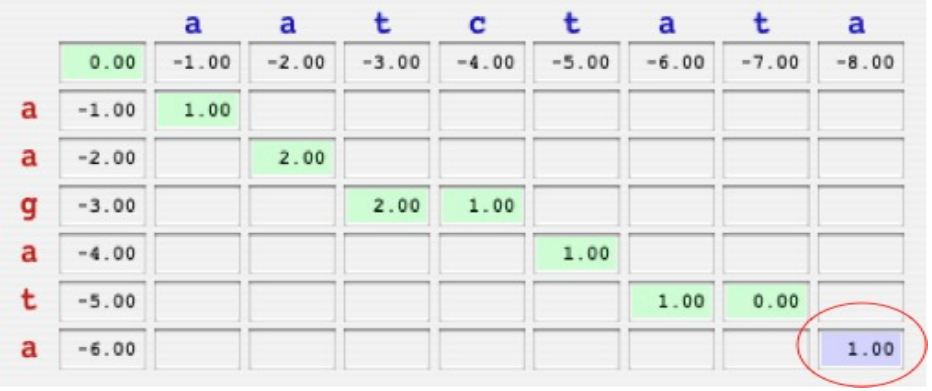
Computing a score

- Click the Reset button
- Use the mouse as above to create the following pattern:

```
aatctata
aag-at-a
```

(recall that this is the first of our example patterns containing gaps)

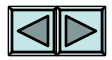
- Now click the Evaluate button in the Control panel, and look at the number at the bottom of the matrix display
- Notice that this is the number we computed as the score for this alignment using the criteria [1,0,-1].



		a	a	t	c	t	a	t	a
	0.00	-1.00	-2.00	-3.00	-4.00	-5.00	-6.00	-7.00	-8.00
a	-1.00	1.00							
a	-2.00		2.00						
g	-3.00			2.00	1.00				
a	-4.00					1.00			
t	-5.00						1.00	0.00	
a	-6.00								1.00

- Now notice that these criteria are specified in the Scoring Panel, on the lower right of the program window.

Gap Penalty	-1.00
Match Score	1.00
Mismatch Score	0.00



Exercise 1.2: Try the rest

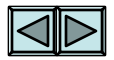
- Click reset, and similarly align the remaining 2 patterns from Exercise 1.1. Record the results.

```
aatctata
aa-g-ata

aatctata
aa--gata
```

- There are a total of 28 patterns in which the query string is extended by 2 gaps to match the length of the target string. Now align each of the remaining 25 patterns. Keep track of the results.

- What is the highest score you get?
  In how many ways can that score be achieved?

Exercise 1.2 Solution



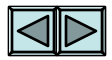
Alot of counting!

- You now know basically how we find the best match.
  - Consider every possible alignment and record the score.
  - Choose an alignment that has the highest score.

- This type of procedure is known as an **optimization problem**.

  In optimization problems there are many potential solutions, however one is interested in finding not any solution, but one that achieves the highest score according to some criteria.

- The difficulties one finds with optimization problems is:
  - Choosing the right scoring criteria
  - Coping with a potentially huge set of cases to consider



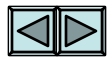
How many matches?

- For now, let's assume we have come up with a good scoring criterion. (We'll worry about that later). How many cases are there to consider?

- Suppose the strings have length *n* and *m* , with $n \geq m$.

- When considering all possible alignments, the total number of strings to be considered is given by the formula

$$\binom{n+m}{m} = \frac{(n+m)\cdot(n+m-1)\cdot\cdots\cdot(n+1)}{m\cdot(m-1)\cdot\cdots\cdot 1}$$



How big does this get?

- In Example 1.2 we considered only 28 candidates. This is because we only placed gaps in the shorter sequence. Considering all sequences, we need to look at

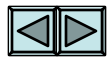
$$\binom{14}{6} = 3003$$

- Matching a pair of sequences of length 20 requires

$$\binom{20}{10} = 184756$$

- Looks like a job for a computer.
  But even a computer might have problems, say, with matching a pair of sequences of length 50:

$$\binom{100}{50} = 100891344545564193334812497256$$



What do we do?

- We have just seen an example of **combinatorial explosion**.
- That is, the number of cases to be considered grows too fast, so that trying to align even moderately long sequences, such as those of length 50, becomes impossible because of too many cases.
- Fortunately, with sequence alignment, there are shortcuts.
- Continued in Part II.

End of Part I

Exercise 1.1 Solutions

a) $1 + 1 + 0 + 0 + 1 + 1 = 4$

b) $1 + 0 + 0 + 0 + 0 + 0 = 1$

c) $0 + 0 + 0 + 1 + 1 + 1 = 3$

d) $1 + 1 + 0 + -1 + 0 + 0 + 0 + 1 = 1$

e) $1 + 1 + -1 + 0 + -1 + 1 + 1 + 1 = 3$

f) $1 + 1 + -1 + -1 + 0 + 1 + 1 + 1 = 3$

Return to Exercise 1.1

Exercise 1.2 Solutions

- aatctata 1
  --aagata
- aatctata 2
  -a-agata
- aatctata 2
  -aa-gata
- aatctata 2
  -aag-ata
- aatctata 1
  -aaga-ta
- aatctata 1
  -aagat-a
- aatctata -1
  -aagata-
- aatctata 2
  a--agata
- aatctata 2
  a-a-gata
- aatctata 2
  a-ag-ata
- aatctata 1
  a-aga-ta
- aatctata 0
  a-agat-a
- aatctata -1
  a-agata-
- aatctata 3
  aa--gata
- aatctata 3
  aa-g-ata
- aatctata 2
  aa-ga-ta
- aatctata 1
  aa-gat-a
- aatctata 0
  aa-gata-
- aatctata 3
  aag--ata
- aatctata 2
  aag-a-ta
- aatctata 1
  aag-at-a
- aatctata 0
  aag-ata-
- aatctata 2
  aaga--ta
- aatctata 1
  aaga-t-a
- aatctata 0
  aaga-ta-
- aatctata 2
  aagat--a
- aatctata 1
  aagat-a-
- aatctata 2
  aagata--

Return to Exercise 1.2